TERRAIN-BASED MEMETIC ALGORITHMS VISUALIZATION

Brad Johnson
B.S., California State University, Sacramento, 2001

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2011
TERRAIN-BASED MEMETIC ALGORITHMS VISUALIZATION

A Project

by

Brad Johnson

Approved by:

________________________________, Committee Chair
Dr. V. Scott Gordon
Date: ______________

________________________________, Committee Member
Dr. Anne-Louise Radimsky
Date: ______________
Student: Brad Johnson

I certify that this student has met the requirements for format contained in the University format manual, and that this Project is suitable for shelving in the Library, and credit is to be awarded for the Project.

_________________________________________________   ___________________
Dr. Nikrouz Faroughi, Graduate Coordinator                   Date

Department of Computer Science
Abstract

of

TERRAIN-BASED MEMETIC ALGORITHMS VISUALIZATION

by

Brad Johnson

In the examination and evolution of genetic algorithms, advancement occurred when utilizing a terrain to model a population of individuals and their associated parameters. This work, detailed by Gordon et al and called TBGA, showed that optimization in traditional CGAs could be improved by this approach. A visualization tool called VisTBGA, developed for viewing the TBGA terrain, was developed for aiding researchers in determining good parameters to utilize with the original CGA.

Later, Azevedo extended the TBGA into a Terrain-Based Memetic Algorithm (TBMA). The TBMA is a diffusion Memetic Algorithm (MA) which utilizes the TBGA concept of spreading parameter values. However, the TBMA's parameter values include local search (LS) step sizes and utilize the 2-dimensional grid for these parameters rather than for evolution parameters.

This project updates the work of VisTBGA to develop a similar visualization tool for the TBMA. The VisTBGA provides visual feedback for researchers, aids with solving problem sets, and provides possibilities for future research into terrain-based approaches to memetic algorithms.

__________________________________________, Committee Chair
Dr. V. Scott Gordon

_____________________
Date

v
TABLE OF CONTENTS

List of Tables ................................................................................................................... viii
List of Figures .................................................................................................................... ix

1. INTRODUCTION .......................................................................................................... 1
   1.1 Evolutionary Algorithms ..................................................................................... 2
   1.2 Genetic Algorithms ............................................................................................. 3
   1.3 Memetic Algorithms ........................................................................................... 4

2. BACKGROUND: TBGA ............................................................................................... 7

3. BACKGROUND: TBMA ............................................................................................. 10
   3.1 mTBMA Definitions, Rules and Pseudocode .................................................... 11

4. BACKGROUND: VECTOR QUANTIZATION ......................................................... 14
   4.1 Vector Quantization and Compression .............................................................. 16
   4.2 Codebook Generation ....................................................................................... 16
   4.3 Vector Quantization Performance Measurement ............................................... 17

5. DESIGN OF VISMTBMA ........................................................................................... 20
   5.1 Motivation.......................................................................................................... 20
   5.2 Methodology and Requirements ........................................................................ 20
   5.3 Migrating Existing mTBMA Codebase to Java.................................................. 21
   5.4 Migration Details ............................................................................................... 22
   5.5 Extending VisTBGA to include migrated mTBMA code .................................... 25
   5.6 Changes to VisTBGA’s Problem Class to Support mTBMA............................ 27
5.7 Changes to other VisTBGA Classes to Support mTBMA ................................. 28
5.8 Extending configuration capabilities of VisTBGA Problem Class .................. 29
5.9 Extending VisTBGA rendering code base for TBMA ................................. 32
5.10 Extending VisTBGA code base to enable mTBMA rendering configuration... 33
5.11 Extending VisTBGA to render mTBMA parameters .................................... 35

6. PRELIMINARY RESULTS ..................................................................................... 38
   6.1 First Experiment – mTBMA with one individual per location .................... 38
   6.2 Second Experiment – mTBMA with 5 individuals per location ................. 42
   6.3 Third Experiment – mTBMA with 20 individuals per location ................ 46
   6.4 Fourth Experiment – mTBMA with New Training Image ...................... 50

7. CONCLUSION AND FUTURE WORK ................................................................. 54

Appendix A Project Source Code ........................................................................ 56
Appendix B Software License ............................................................................. 145

Bibliography ...................................................................................................... Error! Bookmark not defined.
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Average PSNR (dB) values for mTBMA(1)</td>
<td>40</td>
</tr>
<tr>
<td>2.</td>
<td>Percentage of generations with improvement in PSNR mTBMA(1)</td>
<td>41</td>
</tr>
<tr>
<td>3.</td>
<td>Average PSNR (dB) values for mTBMA(5)</td>
<td>44</td>
</tr>
<tr>
<td>4.</td>
<td>Percentage of generations with improvement in PSNR mTBMA(5)</td>
<td>44</td>
</tr>
<tr>
<td>5.</td>
<td>Experiment runtimes of mTBMA(1) and mTBMA(5)</td>
<td>45</td>
</tr>
<tr>
<td>6.</td>
<td>Average PSNR (dB) values for mTBMA(20)</td>
<td>48</td>
</tr>
<tr>
<td>7.</td>
<td>Percentage of generations with improvement in PSNR mTBMA(20)</td>
<td>48</td>
</tr>
<tr>
<td>8.</td>
<td>Runtime of mTBMA across all three experiments</td>
<td>48</td>
</tr>
<tr>
<td>9.</td>
<td>Average PSNR (dB) values for mTBMA with Logo</td>
<td>52</td>
</tr>
<tr>
<td>10.</td>
<td>Percentage of generations with improvement in PSNR for Logo</td>
<td>52</td>
</tr>
<tr>
<td>11.</td>
<td>Runtime of mTBMA across all three experiments</td>
<td>53</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Multimodal Function with Local Optima</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td>Sample TBGA Parameter Distribution</td>
<td>8</td>
</tr>
<tr>
<td>3.</td>
<td>Pseudocode for mTBMA</td>
<td>13</td>
</tr>
<tr>
<td>4.</td>
<td>Sample Voronoi Regions</td>
<td>15</td>
</tr>
<tr>
<td>5.</td>
<td>MSE Equation</td>
<td>18</td>
</tr>
<tr>
<td>6.</td>
<td>Peak Signal to Noise Equation</td>
<td>18</td>
</tr>
<tr>
<td>7.</td>
<td>Sample Image PSNR 45.53dB</td>
<td>18</td>
</tr>
<tr>
<td>8.</td>
<td>Sample Image PSNR 36.81dB</td>
<td>19</td>
</tr>
<tr>
<td>9.</td>
<td>Sample Image PSNR 31.45dB</td>
<td>19</td>
</tr>
<tr>
<td>10.</td>
<td>Original Image</td>
<td>19</td>
</tr>
<tr>
<td>11.</td>
<td>mTBMA Class Relationship Diagram</td>
<td>24</td>
</tr>
<tr>
<td>12.</td>
<td>VisTBGA Class Relationship Diagram</td>
<td>26</td>
</tr>
<tr>
<td>13.</td>
<td>Example Specializations of VisTBGA's Problem Class</td>
<td>27</td>
</tr>
<tr>
<td>14.</td>
<td>Original VisTBGA ConfigPanel Screen</td>
<td>30</td>
</tr>
<tr>
<td>15.</td>
<td>Modified VisTBGA ConfigPanel Screen</td>
<td>31</td>
</tr>
<tr>
<td>16.</td>
<td>Original VisTBGA VisualPanel Screen</td>
<td>33</td>
</tr>
<tr>
<td>17.</td>
<td>VisTBGA VisualPanel Class References</td>
<td>34</td>
</tr>
<tr>
<td>18.</td>
<td>Updated VisTBGA VisualPanel Screen</td>
<td>35</td>
</tr>
<tr>
<td>19.</td>
<td>Sample VisTBGA Rendering with TBMA Parameters</td>
<td>36</td>
</tr>
<tr>
<td>20.</td>
<td>Example VisTBGA Rendering of Terrain after Movement</td>
<td>36</td>
</tr>
<tr>
<td>21.</td>
<td>mTBMA(1) Scale Factors with Lena</td>
<td>39</td>
</tr>
<tr>
<td>22.</td>
<td>mTBMA(5) Scale Factors with Mandrill</td>
<td>39</td>
</tr>
</tbody>
</table>
23. mTBMA( 5 ) Scale Factors with Peppers.................................................................40
24. Original and Resulting Lena Image with mTBMA( 1 ) ..........................................41
25. Original and Resulting Mandrill Image with mTBMA( 1 ) ......................................41
26. Original and Resulting Peppers Image with mTBMA( 1 ) .........................................42
27. mTBMA( 5 ) Scale Factors with Lena .................................................................43
28. mTBMA( 5 ) Scale Factors with Mandrill ...............................................................43
29. mTBMA( 5 ) Scale Factors with Peppers ...............................................................44
30. Original and Resulting Lena Image with mTBMA( 5 ) .............................................45
31. Original and Resulting Mandrill Image with mTBMA( 5 ) ........................................46
32. Original and Resulting Peppers Image with mTBMA( 5 ) .........................................46
33. mTBMA( 20 ) Scale Factors with Lena .................................................................47
34. mTBMA( 20 ) Scale Factors with Mandrill .............................................................47
35. mTBMA( 20 ) Scale Factors with Peppers .............................................................48
36. Original and Resulting Lena Image with mTBMA( 20 ) ...........................................49
37. Original and Resulting Mandrill Image with mTBMA( 20 ) ......................................49
38. Original and Resulting Peppers Image with mTBMA( 20 ) .......................................50
39. mTBMA( 1 ) Scale Factors with Logo .................................................................51
40. mTBMA( 5 ) Scale Factors with Logo .................................................................51
41. mTBMA( 5 ) Scale Factors with Lena .................................................................52
42. Original Logo and results of mTBMA .................................................................53
Chapter 1

INTRODUCTION

Genetic Algorithms (GA) are search algorithms that attempt to loosely model genetic evolution. Practical application of GAs thus far shows their ability to succeed in the realm of optimization problems such as scheduling and shortest path algorithms. Genetic Algorithms depend on simplified notions of selection, crossover, mutation, and survival of the fittest in an artificial population, which models a set of candidate solutions to a problem. The user of the GA is left with the task of selecting the appropriate population size, mutation rate, number of crossover points, fitness function and other appropriate criteria for the problem set.

This effort of manual selection of criteria is significant. Work on the Terrain-Based Genetic Algorithm (TBGA) produced a self-tuning version of a Cellular Genetic Algorithm (CGA). A TBGA contains various combinations of parameter values in different physical locations of a population to form a terrain for solutions to evolve [GPWS99]. Work on TBGA led to a follow-on effort to produce a visualization tool that illustrates the progress of evolution as a gradually evolving terrain map where effective locations of the TBGA show as points with an increasing altitude [GT04].

Most recently, work upon the TBGA lead to a proposal of the Terrain-Based Memetic Algorithm (TBMA) [AG09]. The TBMA is diffusion Memetic Algorithm (MA) that utilizes the TBGA concept of spreading parameter values. However, the TBMA’s parameter values include local search (LS) step sizes and utilize the 2-dimensional grid for these parameters rather than for genetic parameters. The results of the TBMA research indicate a strong case for future work in this research area.
However, limitations exist with the current work with TBMA since there is not a visualization tool to aid researchers developing and utilizing the algorithm. This project proposed to update the work of VisTBGA to develop a visualization tool to promote TBMA, provide immediate feedback for researchers, aid with solving problem sets and ultimately provide possibilities for future research into TBMA. To test the usefulness of the visualization tool, an application or suite of applications will be included within the visualization tool to examine its effectiveness aiding researchers utilizing the applications. Further, the visualization tool will be tested against these problems to determine its applicability in providing insights to TBMA. The visualization tool will aid in the ability to bring an immediate feedback mechanism for measuring the fitness of parameters utilized in the application. The tool will examine methods of demonstrating the success of a population to successfully develop cities, determine mayors and display the results of migration of the population between cities. Details of the operations involved in these steps are included with Chapter 3.

1.1 Evolutionary Algorithms

An understanding of the development of Evolutionary Algorithms (EAs) is provided in many sources such as “Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms” [BÄCK96]. EAs borrow from biological evolution in several ways. EAs include the concepts of reproduction, mutation, recombination and selection. Determination of whether or not an Individual of a population of possible solutions will survive is based upon a fitness function. Evolution takes place after the optimization and fitness functions are applied to the population.
History shows EAs perform well in many areas of study. One advantage of EAs is the algorithms are not tailored to specific problem sets and do not include associated optimizations. This generalization has shown EAs to be effective in many areas of engineering.

However, EAs are subject to difficulties in two primary areas. First, computational complexity is an area of concern. The computational complexity stems from the effort required to perform fitness evaluation. EAs utilize the concept of genotype, or a string representation of the rules that describe an Individual’s genes and phenotype, or the physical representation of the Individual. The second concern occurs since EAs lack a distinction between a genotype and phenotype, which leads to problems during evolution.

1.2 Genetic Algorithms

A primary example of an EA is the Genetic Algorithm (GA). Genetic Algorithms are characterized by a search algorithm that attempts to mimic natural evolution. Typically, the search algorithm is used for optimization problems. As a member of the EA family, they maintain the concepts of inheritance, mutation, selection and crossover.

Genetic Algorithms maintain a population of strings. These strings represent the chromosome or genotype for each possible solution. The solutions are typically referred to as an individual. The algorithm searches for continuous improvement or optimization of a problem by measuring fitness. Each individual’s fitness value measures how closely the string or the chromosome achieves the desired solution for the problem.

For each generation, each individual’s fitness is evaluated. Once completed, individuals are selected from the population, combined, and mutated to form a new population. The updated
population is then the next generation. The process repeats for either a set number of generations or until a specific threshold is met in terms of fitness.

From the process described above, it is somewhat trivial to understand that Genetic Algorithms will have many tunable parameters. The parameters include the number of individuals, mutation rate, crossover rate, and the termination parameters in the form of the number of generations or the fitness threshold. Work in determining how to tune these parameters has been a subject of research for Gordon, Pirie, et al in their work with TBGA [GPWS99]

1.3 Memetic Algorithms

One reference for understanding how Memetic Algorithms (MAs) differ from GAs is found in the work of Moscato and Cotta. Within “A Gentle Introduction to Memetic Algorithms” MAs are explained to differ in their desire to know and exploit as much about the problem as possible [MC03]. MAs expect the parts to communicate prior to the evolution occurring. This communication occurs by including heuristics, local search techniques and many other approaches [AG09].

The algorithm process typically includes three phases, which include selection of existing solutions for reproduction, application of operations to derive new solutions and applying learning to improve solutions. Two distinct forms of Memetic Algorithms exist and are of interest. They are Cellular Memetic Algorithms (CMA) and Diffusion Memetic Algorithms (DMA). CMAs utilize the concepts of GAs and apply knowledge of the problem within their implementation. The CMA applies specific crossover, mutation and local search operators to improve upon the original behavior of CGA. In contrast, the DMA assigns memes or the local
search randomly to individuals. This change permits the individual to utilize the memes from other individuals if their fitness is closer to a solution than its own [BWW00].

Since GAs attempt to follow biological methods, their goal is always to find the best solution to any problem similar to the concepts of survival of the fittest. However, since each GA is implemented without any specific details related to the problem they attempt to optimize, those utilizing GAs face a shortcoming. Many MAs utilize hill climbing to achieve improvements in fitness. Hill climbing will attempt to minimize or maximize a given \( f(x) \). The value of \( x \) is altered during each iteration of the algorithm and the new value of \( f(x) \) is compared to determine if fitness improves. However, it is quite simple for a hill climbing approach to become trapped within a localized area. This leads to the algorithm falling subject to traps of local minima or local maxima. MAs use randomness and biased selection in an attempt to balance diversity with selecting strong strings. Without the combination of randomness and biased selection, the traps of local minima or local maxima become especially problematic in applications that deal with multimodal functions [KPP95]. Multimodal functions are extremely problematic as they contain many local solutions that are not the global optimum.
As shown in Figure 1, algorithms utilizing simple hill climbing along the $f(x)$ values will be subject to many possible failures to find the best solution. The Greedy Algorithms look for the best local solution available at each stage and moves to that individual and are thus subject to being trapped in this manner [FM93] [FR98]. Stochastic Hill Climbing randomizes the neighbor at random and only replaces the individual if enough fitness improvement is achieved. In [KPP95], Kvasnicka explores hill climbing and multimodal functions. Within this work, learning is shown to aid the ability of an algorithm to avoid local minima or local maxima. Memetic Algorithms and local search techniques utilize these optimization techniques and attempt to find global optimized solutions by maintaining diversity while still looking for better solutions.
Chapter 2

BACKGROUND: TBGA

Whether or not an evolutionary algorithm is implemented in a traditional genetic algorithm or with a memetic algorithm, anyone implementing an algorithm must consider the parameters utilized within the solution. While each algorithm may benefit from different values for mutation, crossover, and selection, each implementation will require tuning of the parameters. Fogarty, Schraudolph and Belew, Tsutsui and Fujimoto, Smith and others explored various attempts to self-tune algorithms or alter parameters over time. However, these approaches are complex to implement. A simpler approach for tuning these parameters was introduced by Gordon, Pirie, et al with Terrain Based Genetic Algorithms [GPWS99]. In TBGA, a terrain is formed by evenly distributing parameters along the axes of a population space. Each space in the terrain is given a unique set of parameters, and holds individuals in the population as shown in Figure 2 - Sample TBGA Parameter Distribution.
TBGA utilizes an implementation of a fixed-topology Deme-4 CGA [GMW94]. Each individual is evaluated across each generation, and the mate is selected by evaluating the fitness of each individual north, south, west and east of the current position. The implementation requires crossover to occur always and mutation to be applied to the resulting two individuals. If either of the offspring’s fitness represents an improvement over the original fitness, the appropriate offspring replaces the individual at its space within the terrain.

The work of Gordon, Pirie, et al showed that this approach could and did lead to improved results over the traditional CGA. In their evaluation, multiple algorithms resulted in improved performance when parameters were distributed across the demes. This improvement leads to the possibility of tuning CGAs from the results of the TBGA computations. Unfortunately, the results were not clustered around particular parameters. As a result, Gordon, Pirie, et al utilized a weighted average of the results to feed the original CGA implementations.
In each case, the CGA with the TBGA-tuned parameters outperformed the original CGA implementation and the pure TBGA implementation of the algorithm.
Chapter 3

BACKGROUND: TBMA

Following on to the work of the TBGA, Azevedo and Gordon proposed an algorithm where a memetic algorithm is permitted to benefit from some of the advancements of the TBGA. Azevedo and Gordon note the progress brought to TBGA via the work of Krink and Ursem, called the TB Patchwork Model (TBPM) [KU00]. In this model, individuals within TBGA are permitted to move, and thus more effectively exploit parameters found to be effective for solving the solution at hand. Azevedo and Gordon also note the work of H-cGA by Janson et al [JADM06], in which strings with a higher fitness move toward the center of a changing grid. Each of these forms of genetic algorithms finds its way into TBMA as Azevedo and Gordon propose four variants of the TBMA. The algorithms are the Stationary TBMA, Motioner TBMA, Local Adaptive-sTBMA, and the Hierarchical Adaptive-TBMA.

Stationary TBMA (sTBMA), or the simplest form proposed, closely follows the work of TBGA. Each individual selects a mate from the Deme-4 neighborhood, utilizes the scale factor of the Individual’s position on the terrain and updates the population during each bath.

- The Motioner TBMA (mTBMA) follows the TBPM and adds in the utilization of Accelerated KMeans (AKM) for the local search. This permits individuals to move around a terrain to exploit better parameters. However, in mTBMA evolution occurs via AKM and allows the two best solutions of the three involved in the tournament to survive. Isolated individuals undergo mutation and local search. The new subpopulation undergoes sTBMA to perform evolution and the population is updated with the finalized results.
- The third algorithm, Local Adaptive-sTBMA (LA-sTBMA) changes terrain values at the cell level during local search. It closely follows sTBMA with the exception of performing
an adjustment after a specified number of generations. This permits the population to tune itself to the best parameters from the previous number of generations.

- The final algorithm, Hierarchical Adaptive-TBMA (HA-TBMA) ranks individuals by fitness and allows individuals to compete for locations near the top ranked individual. This permits individuals with similar fitness to mate. However, each cell of the terrain is limited to one individual unlike mTBMA.

The results of the four algorithms found that each outperformed the CMA for the same problem set. Further, Azevedo and Gordon found mTBMA would routinely outperform the remaining three alternative forms. The explanation for the improved performance was “…the reason the mTBMA exhibits superior performance is that it is able to find and exploit dynamically-changing parameter values that correspond effectively with the trajectory that the VQ codebooks happen to be taking through the solution space during evolution” [AG09]. The performance of mTBMA leads to its inclusion within this project. The details of the mTBMA are included here for reference.

3.1 mTBMA Definitions, Rules and Pseudocode

The following definitions and rules are taken from [AG09] and form the basis for the mTBMA approach.

- **Definition 1.** A city C is a collection of $h = 1, \cdots, h_{\text{max}}$ individuals (citizens) which share the same physical location in the terrain. The parameter $h_{\text{max}}$ gives the maximum number of citizens a city can support.

- **Definition 2.** A citizen is an intelligent agent, which, at each generation, performs RULE 1.
• **Rule 1.** If there is a neighbor with better fitness value at city $C_b$ and $|C_b| < h_{max}$, migrate to the city of the best adapted neighbor; else perform RULE 2 with probability $p$.

• **Rule 2.** Migrate to a random distance 1 cell located somewhere in any of the cardinal directions \{N, E, S, W\} and intermediate directions \{NE, SE, SW, NW\}.

• **Definition 3.** A Mayor is the most fit citizen in a city and cannot be replaced by any offspring.
The flow of the mTBMA is summarized in the pseudocode below:

```
INITIALIZE Terrain and Population
WHILE current generation count < max generations
    CountNumberOfCities( Terrain )
    INITIALIZE Mayor Population to Null
    FOR ALL Cities |C_i| IN Terrain
        x_i = getX( |C_i| )
        y_i = getY( |C_i| )
        N_i = getIndividual( |C_i| )
        N_1 = getScaleFactor1( x_i, y_i )
        N_2 = getScaleFactor2( x_i, y_i )
        IF getIndividuals( |C_i| ) == 1 AND N_i is isolated THEN
            LocalSearch( Mutate( N_i ), N_1, N_2 )
            RandomWalk( T, N_i )
        ELSE
            performAKM( CountAt( |C_i| ), N_1, N_2, CountAt( |C_i| ) - 1 )
            M = MAX OF ( previous M.fitness or max( |C_i|.fitness ) )
        END IF
    END FOR
    REMOVE M from Population
    FOR ALL Individuals I_i IN Population
        I_j = FIND MAX FITNESS for I_i’s neighbors ON Terrain
        I_i’ = CROSSOVER( I_i, I_j )
        SET N_1 and N_2 to ScaleFactor values FOR I_i
        performAKM( Mutate( I_i’ ), N_1, N_2 )
        evaluateFitness( I_i’ )
    END FOR
    FOR ALL Individuals I_i IN Population
        IF( Fitness( I_i’ ) ≥ Fitness( I_i ) THEN
            I_i = I_i’
        END IF
    END FOR
END WHILE
```

Figure 3 - Pseudocode for mTBMA
The TBMA has been successfully applied to image compression through the use of vector quantization. The goal of testing TBMA for this application was to confirm if, like the TBGA, the tuning of vector quantization parameters could be improved upon by utilizing a genetic algorithm. This section focuses on the background leading to this approach.

Vector quantization is a technique from signal processing, utilized to provide data compression, and utilized within audio and video compression codecs. To utilize vector quantization, the algorithm maps a series of k-dimensional vectors in a vector space $\mathbb{R}^k$ into a set of vectors $Y = \{y_i, i = 1, 2, \ldots, N\}$. Each resulting vector $y_i$ is called a code word. The overall resulting set of code words makes up a codebook. Finally, the set of nearest neighbors to each $y_i$ is named the Voronoi region.
Figure 4, taken from [MQ11] demonstrates a sample set of codewords. The x locations mark input vectors. The red dots mark code words and the Voronoi regions exist within the boundary lines. For each input vector, the selected code word utilized to represent the input vector exists within the same Voronoi region. The selection of which code word to utilize is based on finding the closest Euclidean distance from the input vector.
4.1 Vector Quantization and Compression

Since TBMA focuses on image compression, vector quantization typically performs compression within images by reducing the color space. The goal is to reduce the total number of colors maintained within the image without compromising image quality [PTAP01]. Since the original image utilizes a large number of input vectors, through replacing these with representations from a codebook, it is quite feasible to reduce the size of the image. However, it is critical to maintain a copy of the codebook to enable decompression of the codebook’s representation of the image.

4.2 Codebook Generation

Codebook generation is typically time consuming and computationally expensive. The designer must initially determine the number of code words and how many to include in a code book. The simplest form flows from the work of Linde, Buzo and Gray [LBG80]. Their work, commonly referred to as the LBG algorithm, is iterative. LBG starts with a given code book to segments a training set through an exhaustive search. Next, the training vector records the index of the closest code word in the codebook. After completing the search, the centroid of the training vectors replaces each code word. This process assures the new code word is the closest of all available code words. The new codebook is utilized in the next iteration to minimize distortion. This process repeats until the difference in distortion is less than a given threshold. Attempts to reduce the computation time required for LBG have resulted in partial distance search (PDS) and triangle inequality elimination rule (TIE) [RP99]. Each of these algorithms has been able to improve upon the performance of LBG while maintaining effective results.
The LBG algorithm is also known by the names of Generalized Lloyd algorithm (GLA) or $K$-means algorithm. A further modification to this algorithm set is the Accelerated $K$-Means algorithm proposed by Lee et al [LBS97]. AKM modifies the original centroid function to not perform an iterative function. Instead, the algorithm utilizes the formula:

$$\text{new codevector} = \text{old codevector} \times \text{scale} \times (\text{new centroid of old codevector})$$

The desire by utilizing this new formula is to speed up the process of finding the direction the code vector should move to lead to convergence [CH98]. The scale factor becomes of great interest in determining just how fast an algorithm will converge and perhaps may be tunable within a genetic algorithm as proposed by Azevedo and Gordon.

4.3 Vector Quantization Performance Measurement

The comparison of any original data source and the result of decompressing the data source after utilizing a lossy compression algorithm is frequently stated utilizing Peak Signal-to-Noise Ratio (PSNR). PSNR refers to the ratio of a maximum possible power of a signal and the corruption of the signal from the noise that impacts the signal. PSNR is typically given in terms of logarithmic decibels.

The difficulty in utilizing PSNR within image compression is the human eye. Perception is not constant and at any given time an item with a lower PSNR may appear closer to the original image than another image with a higher PSNR. To address this concern, Mean Standard Error (MSE) defines the difference between the original image (I) and a second image (K). The formula for MSE is given as:
Within TBMA, gray scale images are utilized giving a maximum value for each pixel of $2^8 - 1$ or 255. This represents the maximum power for each pixel and allows PSNR to be defined as:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_l^2}{MSE}\right)$$

$$= 20 \cdot \log_{10}\left(\frac{MAX_l}{\sqrt{MSE}}\right)$$

**Figure 5 - MSE Equation**

**Figure 6 - Peak Signal to Noise Equation**

The following images demonstrate how JPEG compression quality values impact PSNR.
Figure 8 - Sample Image PSNR 36.81dB

Figure 9 - Sample Image PSNR 31.45dB

Figure 10 – Original Image
Chapter 5
DESIGN OF VisMTBMA

5.1 Motivation

In the work of Azevedo and Gordon on mTBMA, a stated goal for next steps involved applying mTBMA across a larger problem set. As with the migration of TBGA to VisTBGA it seems that a visualization tool would aid that effort. Not only would the visualization tool help reinforce the concepts of mTBMA, but it would also provide a platform for extending the algorithm into other problems and other areas of research.

VisMTBMA extends the VisTBGA into a visualization tool for the TBMA, to provide immediate feedback for researchers, aid with solving problem sets and ultimately provide possibilities for future mTBMA research. The visualization tool also should aid in the ability to bring immediate feedback for the fitness of parameters utilized in the application. The tool reveals how the TBMA develops cities, determines mayors, and it displays the results of migration of the population between cities. The algorithm for these operations is explained within Chapter 3.

5.2 Methodology and Requirements

When beginning any software project, it is crucial to remember the software development process. Embracing and extending a current product does not exempt the developer from practicing proper process. This is true of this work to extend VisTBGA’s capabilities to include support for memetic algorithms.
The requirements for the new tool were listed as follows:

- Migrate existing mTBMA code base from C++ to Java
- Extend VisTBGA to incorporate migrated mTBMA code
- Extend configuration capabilities of VisTBGA to include parameters appropriate for TBMA
- Extend rendering code base to enable plotting parameters appropriate for TBMA
- Permit operator to control whether or not movement code is utilized
- Extend rendering code base to enable displaying cities

5.3 Migrating Existing mTBMA Codebase to Java

As with most projects, the bulleted requirements list implies many sub-requirements. An expanded list would include steps such as confirming the operational status of the original code base, analyzing the code base for data structures unavailable within Java, and examining the code structure for programming concepts in C++ that are unavailable in Java.

Determining the operational status of the original codebase appeared trivial at first. However, since the original development was not with English as the first language, the first barrier was translating Portuguese to English. However, many variable names and debug output was still in Portuguese. The code was also part way through optimizations from Azevedo to the mTBMA code base. Several methods would not compile as a result of these partial changes. Fortunately, the changes were limited in scope and Azevedo was able to quickly confirm the original code base’s intent and the changes required to return to that state.
After the code compiled and test runs of the code were completed, examination of the design and implementation started. There were approximately 2500 lines of code spread across 5 header files and 5 source files. The different files represent different data structures and not necessarily different classes. In fact, the only class container within the code was for implementing a pseudorandom number generator. The remaining items are implemented as structs with inline functions to manipulate the data structures. The code utilizes multiple data structures found in the Standard Template Library (STL) such as vectors, maps and iterators. The code also utilizes common structural components such as typedefs and inline functions. The next step involved examining what elements could migrate unchanged and what would be unsupported within Java.

5.4 Migration Details

NetBeans was selected as the IDE. Next, the process of becoming familiar with the IDE followed implementing several small sample programs. This permitted a non-Java programmer the ability to learn the style and form of Java code, how to utilize classes within Java and some of the caveats C++ programmers must know in Java such as pass by reference versus pass by value. Several web resources were valuable in this process and key to the initial efforts in migrating the mTBMA code [DWHI06] [ZIMA11] [WESL11] [HORS11].

Some expected complications occur with migrating the C++ code. C++ programmers benefit from the ability to utilize unsigned data objects. Unfortunately, within Java the ability to ignore the sign of an object does not exist. This provided many complications for the pseudorandom number generation code. Much of the work involves bit-shifting of variables within that portion of the code base. Confirming that the data structure changes did not
negatively impact the behavior of the codebase was paramount to the success of the code migration.

STL objects such as map and vector are utilized in mTBMA. Since these data structures exist in similar form within Java, albeit with the vector class being deprecated, the decision was made to migrate them unchanged into the Java code. Replacement of stl::map came in the form of Java’s HashMap. As with any map the hash function is key to the success of the map function. In mTBMA, the mapping is typically between an integer and a vector and rarely exceeds 1024 unique hash keys. With this in mind, it was decided to keep the built-in hash function as the risk for collision was low.

Figure 11 represents the class structure utilized in the migration. The random class migrated into its own class RandomMTBMA. The structures for representing the terrain, an individual and the vector quantization implementation became their own classes. Finally, the main structure that contains the actual algorithm implementation for mTBMA became the central class, which utilizes the remaining classes to complete its work.
Figure 11 - mTBMA Class Relationship Diagram
5.5 Extending VisTBGA to include migrated mTBMA code

VisTBGA’s class relationships are shown in Figure 12. The overall concept is similar to the mTBMA implementation. The terrain implementation of mTBMA is contained within VisTBGA’s Population class. The Individual class is almost identical to mTBMA. The differences surround the specific implementation details related to storage of chromosome data. What remains from the mTBMA diagram is the specific implementation of the algorithm itself. Upon a quick examination of VisTBGA, it is clear to determine the abstract class Problem is where additional details of the specific algorithm would reside.
Figure 13 – Example Specializations of VisTBGA’s Problem Class

Figure 13 illustrates the class inheritance where specific algorithm implementations implement specific instances of the Problem class. Embracing and extending the Problem class was the desired approach to implement mTBMA for VisTBGA.

5.6 Changes to VisTBGA’s Problem Class to Support mTBMA

As migration of the mTBMA code started, it was immediately apparent that the Problem class would need to be extended. VisTBGA’s Problem class contains several get and set methods to communicate to the rest of the code base details of the specific algorithm’s implementation details. From the discussion of memetic algorithms, the algorithm needs to allow its local search parameters to be viewed and potentially modified. From the specific problem of mTBMA, it is
known that these parameters would be spread across the population. From Azevedo’s implementation, it is also known that the specific instance of the vector quantization code would need to be available for other classes to utilize. All of these new requirements were met by adding them as abstract methods to the base class Problem.

5.7 Changes to other VisTBGA Classes to Support mTBMA

While implementing the changes to Problem and while migrating the mTBMA code, it became apparent other classes within VisTBGA would also require updates. The Individual class contains multiple get and set methods to retrieve and store implementation specific details for the existing problems. mTBMA requires expanding the Individual class to support its specific requirements. These included extending the class to maintain a copy of the genotype and phenotype for each mTBMA Individual and support setting and retrieving these variables. The Individual class was also extended to support setting and retrieving the Individual’s mean standard error for its specific solution within the terrain. Finally, the ability to store, retrieve and manipulate the quantization error was added.

mTBMA also requires specific changes to the Population class. One of the primary responsibilities of the Population class within VisTBGA is to instantiate each Individual within the problem set. For mTBMA, as Individuals are born, they are given their initial codebook. Since this step is unique to mTBMA the Problem class was extended to allow the class to determine if the specific algorithm requires additional steps unique to mTBMA or other memetic algorithms. After distributing the codebooks, Population also distributes the local search parameters. Population was extended to implement a distribution algorithm similar to VisTBGA’s sift code and called only when implementing a memetic algorithm. Population also
carries the responsibility of performing mutation and crossover within VisTBGA. mTBMA’s implementation of crossover and mutation requires the singleton pseudorandom number generator to exist within the Population class. This fits VisTBGA’s implementation as the population exists throughout the execution of each generation and each experiment. Therefore, Problem implements the mTBMA random class.

Within VisTBGA, Population is the class that results in evolution of the Individuals. Again, this fit well with mTBMA’s view of evolution. Most of mTBMA’s specific evolution code was implemented within VisTBGA’s Population class. This meant VisTBGA’s evolve method was extended to maintain the original genetic algorithm implementations of evolution and the specific requirements for memetic algorithms when the user selects a memetic problem. Population now contains the GENITOR and tournament code from mTBMA. This detail means that the methods for determining the most fit neighbors also migrate within the Population class. Finally, all mTBMA movement code exists within the Population class as this is where VisTBGA maintains a view of where Individual’s exist in the terrain.

5.8 Extending configuration capabilities of VisTBGA Problem Class

VisTBGA allows the user to specify specific configuration details appropriate for genetic algorithms as shown in Figure 14. Memetic algorithms have additional things the user must be able to specify. mTBMA also adds in specific details and requirements for customization at runtime. From Azevedo’s work, VisTBGA must therefore be extended to allow specifying the local search parameters, specifying the image to utilize for compression and decompression, and the ability to enable or disable the movement aspect of mTBMA from our requirements for this project.
Figure 14 - Original VisTBGA ConfigPanel Screen
Figure 15 shows the resulting changes to the ConfigPanel class. The new parameter controls are only enabled when the user selects a memetic algorithm. The Problem class was further extended to allow the specific implementation to specify the labels for the algorithm’s local search parameters (shown above as Lower Bound and Upper Bound).
5.9 Extending VisTBGA rendering code base for TBMA

VisTBGA’s VisualPanel class renders the screen shown in Figure 15. While the details were appropriate for the implementation of TBGA, items needed to be changed for the mTBMA. Primarily, the configuration options do not include any of the options of importance to visualizing an implementation of mTBMA. From the knowledge gathered reviewing Azevedo’s work, it is apparent the rendering of the Individuals would differ. No longer would the interest be on crossover and mutation with mTBMA. Instead, the interest would focus around the local search parameters. The rendering must also include the ability to represent the cities, or the movements of the Individuals, to aid the user in determining what parameters are being utilized for evolution. Both are similar in their nature to what is plotted within VisTBGA and require minor changes to the code.
5.10 Extending VisTBGA code base to enable mTBMA rendering configuration

From the class relationship shown in Figure 17 it is apparent which classes will require updates to support the new configuration items. As mentioned within the previous section, VisualPanel is responsible for rendering all configuration aspects of how the experiments are rendered. It has ties to the Rendering and Display classes. These ties and their associated get and set methods permit the change in behavior needed to highlight the details of interest within each problem and algorithm analysis during each run. By adding new parameters and control logic to VisualPanel, the desired behavior can be achieved.
Figure 17 - ViSTBGA VisualPanel Class References
Figure 18 - Updated VisTBGA VisualPanel Screen

Figure 18 shows the final changes to VisualPanel to support mTBMA.

The new options are:

- **Show Scale Factor**
  
  When selected, and only available when utilizing a memetic algorithm, this instructs the renderer to display the local search parameters instead of the genetic algorithm parameters for crossover and mutation.

- **Show City**
  
  When selected, and only available when utilizing a memetic algorithm, this instructs the renderer to display the current population distribution rather than the number of times an Individual was selected as the most fit solution.

### 5.11 Extending VisTBGA to render mTBMA parameters

Now that the user can specify the desired behavior, the VisTBGA code needed to allow the rendering code to retrieve the appropriate parameters and display them. As shown in Figure 19, displaying the local search parameters, or the scale factors for mTBMA, requires the rendering code to retrieve these parameters from the terrain. To accomplish this goal, Renderer was extended to permit VisualPanel to control whether or not to plot the local search parameters. When local search is enabled, the rendering code retrieves the local search parameters from its
reference to the Problem instance. It then, as with the original VisTBGA code, determines the appropriate points to fill in the labels and plots those values along the axis.

![Figure 19 - Sample VisTBGA Rendering with TBMA Parameters](image1)

![Figure 20 - Example VisTBGA Rendering of Terrain after Movement](image2)

Now that the rendering code can display the appropriate data labels, the focus switches to plotting the location of each individual within the terrain. VisTBGA maintains a view of the population within the Display class. Here, the analysis code aids Display in determining the
weighted average, peak point and the number of times each individual was selected as the most fit individual. Modifying Display to also track the number of individuals at each location was selected over enabling this at the Renderer class to avoid adding unneeded coupling and cohesion between the classes.

The analysis code is called at the end of each generation. The analysis code determines if the peak point, average point or if individuals moved. After this data is determined, the ConfigPanel data points are updated along with the Display data points. At the next display update the plot represents the result of the generation.
Chapter 6
PRELIMINARY RESULTS

6.1 First Experiment – mTBMA with one individual per location

The initial experiments attempt to reproduce the findings related to mTBMA in Azevedo’s work. VisMTBMA was configured to match the parameter set listed by Azevedo. This means the population size was set to 25, or a grid of 5x5. Further, the scale factors were evenly distributed across the population with a range of 1.0 to 1.4, mutation was set to occur 8% of the time, and the Lena, peppers and Mandrill images were utilized. However, the movement code of mTBMA was disabled for these runs. The desire was to see PSNR worsen from the findings to validate the necessity for movement. Additionally, this permits an opportunity to confirm if movement impacts the scale factor selection and migration around the available scale factor settings as implied within Azevedo and Gordon’s work.

Figure 21 through Figure 23 show the resulting selection of scale factors across the experiment for the three images.
Figure 21 - mTBMA(1) Scale Factors with Lena

Figure 22 - mTBMA(5) Scale Factors with Mandrill
Table 1 summarizes the PSNR values and range for the experiments. There is quite a difference between the results Azevedo found and the run with the ported data. Lena shows an improvement of 7.73 dB and Mandrill shows the most improvement with 13.44 dB. The other point of interest is the increased variance in the output. With a maximum of one Individual per cell, the algorithm is forced to evaluate each individual. This appears to result in further improvements in PSNR. The increased variance could indicate additional experiments are required to allow the results to converge.
The following table indicates the number of times throughout the experiments where an improvement was made upon PSNR. With the exception of Lena, improvements tend to occur regularly throughout the experiment’s run.

<table>
<thead>
<tr>
<th>mTBMA(1)</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Generations with PSNR improvements</td>
<td>3.85%</td>
<td>12.6%</td>
<td>11.5%</td>
</tr>
</tbody>
</table>

Table 2 – Percentage of generations with improvement in PSNR mTBMA(1)

The following figures show the original image and the resulting image after the experiment.

Figure 24 - Original and Resulting Lena Image with mTBMA(1)

Figure 25 - Original and Resulting Mandrill Image with mTBMA(1)
6.2 Second Experiment – mTBMA with 5 individuals per location

Given the results of the first experiment and following Azevedo’s experimenting with movement, a second experiment is performed. The second experiment retains the same parameter settings for the algorithm tuning. However, motion is now enabled and a maximum value of 5 individuals per position is set. The PSNR was expected increase as the individuals migrate and find improved solutions. The following figures again summarize the selection of scale factors across the experiment.
Figure 27 - mTBMA(5) Scale Factors with Lena

Figure 28 - mTBMA(5) Scale Factors with Mandrill
Again, the PSNR is drastically improved as shown in Table 3 and Table 4. However, we now see a marked improvement with the number of generations reporting improvements. This is especially true with Lena as now over 10% of the generations result in improvement in PSNR values. Unfortunately, the variance between generations is still larger than Azevedo and Gordon’s work. And, we do not see marked improvement in PSNR. This may indicate we are reaching the limit on PSNR when utilizing a codebook size of 256.

<table>
<thead>
<tr>
<th>Image</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTBMA( 5 )</td>
<td>38.68 ± 0.16</td>
<td>39.03 ± 0.28</td>
<td>38.99 ± 0.3</td>
</tr>
<tr>
<td>mTBMA Azevedo</td>
<td>31.03 ± 0.04</td>
<td>25.58 ± 0.01</td>
<td>30.83 ± 0.03</td>
</tr>
</tbody>
</table>

Table 3 - Average PSNR (dB) values for mTBMA( 5 )

<table>
<thead>
<tr>
<th>Image</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTBMA( 5 )</td>
<td>10.1%</td>
<td>13.8%</td>
<td>12.4%</td>
</tr>
</tbody>
</table>

Table 4 – Percentage of generations with improvement in PSNR mTBMA( 5 )
One new measurement to track is runtime of the algorithm. Between mTBMA(1) and mTBMA(5), the total runtime required for each image reduced significantly. This is a direct result of the improved behavior of mTBMA when cities form. The algorithm required fewer offspring when evaluating the cities, which drastically reduces the CPU time required to perform evolution. This reduction in CPU time is shown in Table 5. For comparison sake, the image with the highest PSNR across all generations is shown below.

<table>
<thead>
<tr>
<th>Experiment / Runtime</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTBMA(1)</td>
<td>183 minutes</td>
<td>190 minutes</td>
<td>175 minutes</td>
</tr>
<tr>
<td>mTBMA(5)</td>
<td>70 minutes</td>
<td>71 minutes</td>
<td>69 minutes</td>
</tr>
</tbody>
</table>

Table 5 – Experiment runtimes of mTBMA(1) and mTBMA(5)

Figure 30 - Original and Resulting Lena Image with mTBMA(5)
6.3 Third Experiment – mTBMA with 20 individuals per location

The third experiment extends the number of individuals to 20 per position. The intent in increasing the number of individual per location is to stress the application of rule 1 and rule 2 as described within Chapter 3. That is to say, individuals will be far more likely to either move to a city or move away based on fitness within the city. This movement will permit both the local search and the AKM to run frequently. The other benefit is that as larger cities form the computation time should decrease unless there are always isolated individuals, for the same reasons as explained in the second experiment. If runtime decreases and similar PSNR values are returned this could provide an optimization for VQ. However, close attention will be paid to
scale factor to make sure the mTBMA behavior with regard to scale factor selection is not drastically changed.

Figure 33 - mTBMA(20) Scale Factors with Lena

Figure 34 - mTBMA(20) Scale Factors with Mandrill
Figure 35 - mTBMA(20) Scale Factors with Peppers

<table>
<thead>
<tr>
<th>Image</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTBMA(20)</td>
<td>38.67 ± 0.18</td>
<td>38.99 ± 0.30</td>
<td>40.42 ± 0.57</td>
</tr>
<tr>
<td>mTBMA Azevedo</td>
<td>31.03 ± 0.04</td>
<td>25.58 ± 0.01</td>
<td>30.83 ± 0.03</td>
</tr>
</tbody>
</table>

Table 6 - Average PSNR (dB) values for mTBMA(20)

<table>
<thead>
<tr>
<th>Experiment / Runtime</th>
<th>Lena</th>
<th>Mandrill</th>
<th>Peppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTBMA(1)</td>
<td>183 minutes</td>
<td>190 minutes</td>
<td>175 minutes</td>
</tr>
<tr>
<td>mTBMA(5)</td>
<td>70 minutes</td>
<td>71 minutes</td>
<td>69 minutes</td>
</tr>
<tr>
<td>mTBMA(20)</td>
<td>66 minutes</td>
<td>69 minutes</td>
<td>67 minutes</td>
</tr>
</tbody>
</table>

Table 8 – Runtime of mTBMA across all three experiments

Unfortunately, as the results above show, adding more individuals per cell does not appear to improve the performance. With Lena still showing results in the 38.6 dB range across all three experiments it seems the results are being limited more by the
ability of the codebooks to represent the image rather than the ability of the algorithm to improve results. Further, the runtime duration was almost identical showing this did not provide an improvement in runtime behavior either. Figure 36, Figure 37, and Figure 38 show the resulting images.

Figure 36 - Original and Resulting Lena Image with mTBMA( 20 )

Figure 37 - Original and Resulting Mandrill Image with mTBMA( 20 )
6.4 Fourth Experiment – mTBMA with New Training Image

The fourth experiment attempts to confirm the behavior of mTBMA by adding a new image. The image follows the same guidelines as expressed by Azevedo and Gordon for images utilized in mTBMA as the image is 256x256 and gray scale. The same codebook generation code used to generate the Lena, Mandrill and Peppers codebooks produced the codebook for Logo. The results of the scale factor selection is shown in Figure 39, Figure 40 and Figure 41.
Figure 39 - mTBMA(1) Scale Factors with Logo

Figure 40 - mTBMA(5) Scale Factors with Logo
It is clear to see the selection behavior for scale factor remains unchanged with the new image. The question remains as to how the algorithm performed with regard to PSNR, runtime and number of generations with improvements.

<table>
<thead>
<tr>
<th>Image</th>
<th>mTBMA(1)</th>
<th>mTBMA(5)</th>
<th>mTBMA(20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logo</td>
<td>43.21 ± 1.64</td>
<td>43.07 ± 1.58</td>
<td>42.97 ± 1.59</td>
</tr>
</tbody>
</table>

Table 9 - Average PSNR (dB) values for mTBMA with Logo

<table>
<thead>
<tr>
<th>% Generations with PSNR improvements</th>
<th>mTBMA(1)</th>
<th>mTBMA(5)</th>
<th>mTBMA(20)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16%</td>
<td>15.7%</td>
<td>12.1%</td>
</tr>
</tbody>
</table>

Table 10 – Percentage of generations with improvement in PSNR for Logo
Table 11 – Runtime of mTBMA across all three experiments

The increased PSNR and variance indicates the number of generations for this experiment needs to be increased. The assumption here is the relative simplicity of the image permitted improvements over the more complex images utilized by Azevedo. The resulting images are shown below for each mTBMA run.

Figure 42 - Original Logo and results of mTBMA

The images above are in the order of original image in the top left, mTBMA( 1 ) in the top right, mTBMA( 5 ) on the bottom left and mTBMA( 20 ) on the bottom right.
Chapter 7

CONCLUSION AND FUTURE WORK

The stated goals for VisMTBMA are complete. The original mTBMA code base was successfully migrated to Java. This migrated mTBMA was successfully incorporated within the VisTBGA framework. VisTBGA was then extended to support configuration of the algorithm, controlling whether movement is enabled and adding rendering of the mTBMA parameters rather than CGA parameters. These extensions were able to show a successful implementation of mTBMA within VisMTBMA through reproducing the work of Azevedo. Those utilizing the tool can easily see how the parameters migrate and flow through the scale factors. The tool also reinforces the usefulness of scale factors in the range of 1.0 to 1.4.

Unfortunately, during the project timeline, it was not possible to add additional applications and local search implementations to VisMTBMA. As a result, it is unclear if the framework will stand up to the demands of adding additional problems. The configuration requirements for these additional algorithms may require additional changes to the VisMTBMA framework.

Throughout the implementation of VisMTBMA, several items for future work revealed themselves. Due to speed optimizations lost in the transition from C++ to Java, optimizations are likely available to reduce the computation time required within the tool. From the original codebase, it is apparent that there are limitations occurring that might be improved upon through JRE enhancements, code changes or data structure changes. Further, the codebooks and images must be available to the tool prior to runtime and require code changes to migrate. Ideally,
VisMTBMA should read the list of available images dynamically and generate the appropriate codebooks if they are not already present.

The dynamic nature of the PSNR values returned throughout the experiments also indicates additional work should be performed after the performance of VisMTBMA is enhanced. Increasing the codebook size, the terrain size and the number of generations would aid future research as to the cause for the variance. The additional computation time for the larger codebook may further show improvements related to the number of individuals at each location and the associated runtime.

Finally, as stated by Azevedo and Gordon, only when additional local search algorithms are tested within mTBMA will the usefulness of mTBMA and VisMTBMA be fully known. VisMTBMA will be a success when it is shown to aid researchers in optimizing the parameters for a larger set of applications.
APPENDIX A
Project Source Code

The Visual Terrain-Based Genetic Algorithm system [GT04] is a large project, available from the Internet in both executable binary and source code form at http://gaia.ecs.csus.edu/~evolves at the time of this writing. As such, only files that underwent changes during this study to upgrade the applet's capabilities are published here. It is written in Java, and should be compiled with Sun Java Development Kit 1.6 or higher, and used with Sun Java Virtual Machine version 1.6 or higher. It may or may not compile or execute correctly with earlier versions of either the Java Development Kit or Java Virtual Machine, or alternative implementations of the Java system.

A.1 – Changes to Analysis.java

/**
 * Determines the current weighted average and most frequent point on
 * the visualization graph.
 */
public void analyzeGeneration(Individual genBest) {
    int col, row;
    int idx = genBest.getIndex();
    int sid = problem.getSideLength();

    // Update Peak Information
    dataPeak[idx]++;

    // Determine New Peak
    if (dataPeak[idx] > dataPeak[idxPeak]) {
        idxPeak = idx;
    }

    // Increment Count of Generation Data
    cntGenData++;

    // Decode Column / Row Positions from Index Position
    col = idx % sid;
    row = (idx - col) / sid;

    // Update Weights
    weightCrossover += problem.getCrossoverPoints(col);
    weightMutation += problem.getMutationRate(row);

    // Calculate Weighted Averages
    if (cntGenData > 0) {
        weightAvgCrossover = weightCrossover / (double) cntGenData;
        weightAvgMutation = weightMutation / (double) cntGenData;
    }
if (problem.getMemeticMode()) {  
    // Problem is mtbma, output the new resulting image  
    // Iterate through all positions in the phenotype, replace the appropriate  
    // locations with the codebook entry and output as ascii for each pixel  
    // Makes this compatible with dattopgm executable of Azevedo's.  
    String outputName = "Generation-" + cntCurrentGeneration + ",-" + genBest.getpsnr() + 
    
    try {  
        // Note the 21845 - or 65535/3. If the image size changes, this value needs to change  
        BufferedWriter out = new BufferedWriter(new FileWriter(outputName));  
        ArrayList<ArrayList<Float>> geno = genBest.getGenotype();  
        Map<Integer, ArrayList<Integer>> map = genBest.getPhenotype();  
        ArrayList<ArrayList<Float>> big = new ArrayList<ArrayList<Float>>();  
        for (int k = 0; k < 21845; ++k) {  
            big.add(new ArrayList<Float>(Collections.nCopies(3, new Float(0.0f))));  
        }  
        // Now find the maps and get the appropriate member from the codebook  
        for (int a = 0; a < map.size(); ++a) {  
            if (map.containsKey(a)) {  
                ArrayList<Integer> positions = map.get(a);  
                for (int b = 0; b < positions.size(); ++b) {  
                    // map first is genotype entry  
                    // map second is list of positions in the image  
                    // need to put in all three values for rgb  
                    int position = positions.get(b);  
                    ArrayList<Float> rgb = geno.get(a);  
                    big.set(position, rgb);  
                }  
            }  
        }  
        // Now output the individual pixels  
        for (int i = 0; i < big.size(); ++i) {  
            ArrayList<Float> inner = big.get(i);  
            for (int j = 0; j < inner.size(); ++j) {  
                out.write(inner.get(j) + 
            }  
        }  
        // Make sure to close the file to save it  
        out.close();  
    } catch (IOException e) {  
        System.out.println("Exception writing " + outputName);  
    }  
}  
}  

public void examineMovement(Population pop) {  
    // always called - increment our current generation count  
    +cntCurrentGeneration;  
    cityPeak = pop.examineCity();  
    // For mtbma, output the new population information - similar to output  
    // from original Azevedo code. Helps in debugging and should really only  
    // be run if the user requests the debug output.  
    if (problem.getMemeticMode()) {  
        for (int i = 0; i < problem.getPopulationSize(); ++i) {  
            System.out.print(cityPeak[i] + " ");  
            if ((i % problem.getSideLength()) == problem.getSideLength() - 1) {  
                System.out.println();  
            }  
        }  
    }  
}
private final static String problemList[] = {"Decep3", "Decep4", "F1", "F2", "F3", "F4", "FS", 
"Rastrigin", "Griewangk", "Schwefel", 
"Knap20", "Knap80", "VisMTBMA"};
//Internal list of supported images for mtbma. Add to this if adding a new image
private final static String figureList[] = {"lena", "logo", "mandrill", "peppers"};
/**
 * Initializes configuration panel to default state.
 */
public ConfigPanel(Problem problem, VisualPanel thePanel) {
    this.problem = (Problem) problem.clone();
    this.visPanel = thePanel;
    setBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createTitledBorder("Genetic Parameters"),
        BorderFactory.createEmptyBorder(10, 10, 10, 10)));
    this.setLayout(new GridLayout(2, 1, 0, 0));
    JPanel subPanelA = new JPanel();
    JPanel subPanelB = new JPanel();
    subPanelA.setLayout(new GridLayout(2, 2, 20, 0));
    subPanelB.setLayout(new GridLayout(1, 2, 20, 0));
    Box hb;   // For Horizontal Boxes
    Box vb;   // For Vertical Boxes
    // For Detailed Layout
    GridBagConstraints c;
    GridBagLayout gb;
    JLabel l;
    JPanel p;

    //**************************************************************************
    // Problem JComboBox
    //**************************************************************************
    vb = Box.createVerticalBox();
    //vb.add(Box.createVerticalGlue());
    hb = Box.createHorizontalBox();
    l = new JLabel("Problem: ");
    l.setToolTipText("Select a problem to configure and solve.");
    cmbProblem = new JComboBox(problemList);
    cmbProblem.setMaximumSize(new Dimension(
        cmbProblem.getMaximumSize().width, 
        cmbProblem.getMinimumSize().height));
    cmbProblem.setToolTipText(l.getToolTipText());
    cmbProblem.addActionListener(this);
    l.setLabelFor(cmbProblem);
    hb.add(l);
    hb.add(cmbProblem);
    vb.add(hb);
    subPanelA.add(vb);
// Problem Description

vb = Box.createVerticalBox();
vb.add(Box.createVerticalGlue());
txtaProbDesc = new JTextArea(problem.description());
txtaProbDesc.setEditable(false);
txtaProbDesc.setLineWrap(true);
txtaProbDesc.setWrapStyleWord(true);
txtaProbDesc.setBackground((Color) UIManager.get("Label.background"));
txtaProbDesc.setForeground((Color) UIManager.get("Label.foreground"));
txtaProbDesc.setFont((Font) UIManager.get("Label.font"));

vb.add(txtaProbDesc);
vb.add(Box.createVerticalGlue());

subPanelA.add(vb);

// Random Seed

p = new JPanel();
p.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
sp = new GridBagLayout();
p.setLayout(sp);

c = new GridBagConstraints();
c.anchor = GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 2;
c.gridy = 0;
c.weightx = 1.0;

chkRandomSeed = new JCheckBox("Specify Random Seed");
chkRandomSeed.setSelected(false);
chkRandomSeed.setToolTipText("Specify the seed for the random number generator.");
chkRandomSeed.addItemListener(this);
sp.setConstraints(chkRandomSeed, c);
p.add(chkRandomSeed);

l = new JLabel("Random Seed: ");
l.setToolTipText(chkRandomSeed.getToolTipText());
ftxtRandomSeed = new JFormattedTextField( NumberFormat.getNumberInstance());
l.setLabelFor(ftxtRandomSeed);
c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridx = 1;
c.gridy = 1;
c.weightx = 0.0;
sp.setConstraints(l, c);
p.add(l);

ftxtRandomSeed.setEnabled(false);
ftxtRandomSeed.setFocusLostBehavior(JFormattedTextField.COMMIT OR REVERT);
ftxtRandomSeed.setValue(new Long(problem.getSeed()));
ftxtRandomSeed.addPropertyChangeListener(this);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 1;
c.weightx = 1.0;
gb.setConstraints(ftxtRandomSeed, c);
p.add(ftxtRandomSeed);

subPanelA.add(p);

////////////////////////////////////////////////////////////////////////

// Undefineable Genetic Parameters
////////////////////////////////////////////////////////////////////////

hb = Box.createHorizontalBox();
vb = Box.createVerticalBox();
DecimalFormat f = new DecimalFormat();
f.setDecimalFormatSymbols(new DecimalFormatSymbols(Locale.US));
f.setMinimumIntegerDigits(1);
f.setMaximumFractionDigits(4);
f.setMinimumFractionDigits(4);
f.setGroupingUsed(false);

lblChromosome = new JLabel("Chromosome Length: "+
problem.getChromosomeLength());
lblChromosome.setAlignmentX(Component.LEFT_ALIGNMENT);
lblChromosome.setToolTipText("The length of an individual solution's string.");
vb.add(lblChromosome);

lblPopulation = new JLabel("Population Size: "+
problem.getPopulationSize());
lblPopulation.setAlignmentX(Component.LEFT_ALIGNMENT);
lblPopulation.setToolTipText("The number of individuals in the population.");
vb.add(lblPopulation);

lblTarget = new JLabel("Target Value: "+
f.format(problem.getTarget()));
lblTarget.setToolTipText("The value that solution fitnesses are 
compared against to determine which individual is best.");
vb.add(lblTarget);

////////////////////////////////////////////////////////////////////////

hb.add(vb);
hb.add(Box.createHorizontalGlue());
subPanelA.add(hb);

////////////////////////////////////////////////////////////////////////

// Defineable Genetic Parameters
////////////////////////////////////////////////////////////////////////

p = new JPanel();
p.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
gb = new GridBagLayout();
p.setLayout(gb);

////////////////////////////////////////////////////////////////////////

// Terrain Side Length
////////////////////////////////////////////////////////////////////////
l = new JLabel("Terrain Side Length: ");
l.setToolTipText("The side length for the terrain grid. Population "+ "size is this value squared.");
ftxtSideLength = new JFormattedTextField(NumberFormat.getNumberInstance());
l.setLabelFor(ftxtSideLength);

c = new GridBagConstraints();
c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 0;
c.weightx = 0.0;
gb.setConstraints(l, c);
p.add(l);

ftxtSideLength.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtSideLength.setValue(new Integer(problem.getSideLength()));
ftxtSideLength.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 0;
c.weightx = 1.0;
gb.setConstraints(ftxtSideLength, c);
p.add(ftxtSideLength);

////////////////////////////////////////////////////////////////////////
// Experiments
////////////////////////////////////////////////////////////////////////

l = new JLabel("Experiments: ");
l.setToolTipText("The number of experiments to perform.");
ftxtExperiments = new JFormattedTextField(NumberFormat.getNumberInstance());
l.setLabelFor(ftxtExperiments);

c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 1;
c.weightx = 0.0;
gb.setConstraints(l, c);
p.add(l);

ftxtExperiments.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtExperiments.setValue(new Integer(problem.getExperiments()));
ftxtExperiments.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 1;
c.weightx = 1.0;
gb.setConstraints(ftxtExperiments, c);
p.add(ftxtExperiments);

////////////////////////////////////////////////////////////////////////
// Generations
////////////////////////////////////////////////////////////////////////

l = new JLabel("Generations: ");
l.setToolTipText("The number of generations per experiment.");
ftxtGenerations = new JFormattedTextField(NumberFormat.getNumberInstance());
l.setLabelFor(ftxtGenerations);

c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 2;
c.weightx = 0.0;
gb.setConstraints(l, c);
p.add(l);

ftxtGenerations.setFocusLostBehavior(
  JFormattedTextField.COMMIT_OR_REVERT);
ftxtGenerations.setValue(new Integer(problem.getGenerations()));
ftxtGenerations.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 2;
c.weightx = 1.0;
gb.setConstraints(ftxtGenerations, c);
p.add(ftxtGenerations);

////////////////////////////////////////////////////////////////////////
subPanelB.add(p);

/*
 * Next section adds in the appropriate parameters for mtbma
 * Goal is this is generic enough for all ma's assuming they use two
 * parameters for their local search. Obviously, image would not be
 * used in all cases. But, the other parameters could easily be
 * utilized. Values set here will be passed into problem later.
 * 
 * In general, all of these items should remain disabled until an
 * algorithms that utilizes them is selected.
 */

////////////////////////////////////////////////////////////////////////
// Figure Combo
////////////////////////////////////////////////////////////////////////

p = new JPanel();
p.setBorder(BorderFactory.createEmtyBorder(5, 5, 5, 5));
gb = new GridBagLayout();
p.setLayout(gb);

lblFigure = new JLabel("Figure: ");
lblFigure.setToolTipText("Select a figure to utilize.");
cmbFigure = new JComboBox(figureList);
cmbFigure.setMaximumSize(new Dimension(
cmbFigure.getMaximumSize().width,
cmbFigure.getMinimumSize().height));
cmbFigure.setToolTipText(l.getToolTipText());
cmbFigure.addActionListener(this);
lblFigure.setLabelFor(cmbFigure);

c = new GridBagConstraints();
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 0;
c.weightx = 0.0;
gb.setConstraints(lblFigure, c);
p.add(lblFigure);

cmbFigure.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 0;
c.weightx = 1.0;
gb.setConstraints(cmbFigure, c);
p.add(cmbFigure);

lblFigure.setEnabled(false);
cmbFigure.setEnabled(false);

////////////////////////////////////////////////////////////////////////

// Lower Range for Scale Factor
////////////////////////////////////////////////////////////////////////

lblFactor1 = new JLabel("Factor1:");
//Note this should be generalized to allow problem to set the value
lblFactor1.setToolTipText("The lower bound for scale factor.");
ftxtFactor1 = new JFormattedTextField(NumberFormat.getNumberInstance());
lblFactor1.setLabelFor(ftxtFactor1);
c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 1;
c.weightx = 0.0;
gb.setConstraints(lblFactor1, c);
p.add(lblFactor1);
ftxtFactor1.setEnabled(true);
ftxtFactor1.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtFactor1.setValue(new Float(problem.getLowerScaleFactor()));
ftxtFactor1.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 1;
c.weightx = 1.0;
gb.setConstraints(ftxtFactor1, c);
p.add(ftxtFactor1);
ftxtFactor1.setEnabled(false);

////////////////////////////////////////////////////////////////////////

// Upper Range for Scale Factor
////////////////////////////////////////////////////////////////////////

lblFactor2 = new JLabel("Factor2:");
//Note this should be generalized to allow problem to set the value
lblFactor2.setToolTipText("The upper bound for scale factor.");
ftxtFactor2 = new JFormattedTextField(NumberFormat.getNumberInstance());
lblFactor2.setLabelFor(ftxtFactor2);
c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 2;
c.weightx = 0.0;
gb.setConstraints(lblFactor2, c);
p.add(lblFactor2);
ftxtFactor2.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtFactor2.setValue(new Float(problem.getUpperScaleFactor()));
ftxtFactor2.addPropertyChangeListener(this);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 2;
c.weightx = 1.0;
gb.setConstraints(ftxtFactor2, c);
p.add(ftxtFactor2);

ftxtFactor2.setEnabled(false);

////////////////////////////////////////////////////////////////////////
// Specify if individuals may move or not
////////////////////////////////////////////////////////////////////////
c = new GridBagConstraints();
c.anchor = GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 2;
c.gridy = 3;
c.weightx = 1.0;
chkMovementEnabled = new JCheckBox("Enable Movement");
chkMovementEnabled.setToolTipText("Permit the Individuals to form cities.");
chkMovementEnabled.setSelected(false);
chkMovementEnabled.addItemListener(this);
gb.setConstraints(chkMovementEnabled, c);
p.add(chkMovementEnabled);

////////////////////////////////////////////////////////////////////////
// Maximum number of individuals per location
////////////////////////////////////////////////////////////////////////
l = new JLabel("Max per location: ");
l.setToolTipText("Specifies the maximum individuals per location.");
ftxtMaxPerCell = new JFormattedTextField(NumberFormat.getNumberInstance());
l.setLabelFor(ftxtMaxPerCell);
c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 4;
c.weightx = 0.0;
gb.setConstraints(l, c);
p.add(l);

ftxtMaxPerCell.setEnabled(false);
ftxtMaxPerCell.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtMaxPerCell.setValue(new Integer(problem.getMaxPerCell()));
ftxtMaxPerCell.addPropertyChangeListener(this);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 4;
c.weightx = 1.0;
gb.setConstraints(ftxtMaxPerCell, c);
p.add(ftxtMaxPerCell);

ftxtMaxPerCell.setEnabled(false);
chkMovementEnabled.setEnabled(false);
subPanelB.add(p);

////////////////////////////////////////////////////////////////////////
// Terrain Options
////////////////////////////////////////////////////////////////////////
p = new JPanel();
p.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
gb = new GridBagLayout();
p.setLayout(gb);

c = new GridBagConstraints();
c.anchor = GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 2;
c.gridy = 0;
c.weightx = 1.0;

chkTerrainMode = new JCheckBox("Specify Terrain Parameters");
chkTerrainMode.setToolTipText("Manually configure genetic parameters."
 + " The algorithm will run as a simple cellular genetic algorithm.");
chkTerrainMode.setSelected(false);
chkTerrainMode.addItemListener(this);

gb.setConstraints(chkTerrainMode, c);
p.add(chkTerrainMode);

////////////////////////////////////////////////////////////////////////
// Crossover Points
////////////////////////////////////////////////////////////////////////

l = new JLabel("Crossover Points: ");
l.setToolTipText("The number of points to randomly select along "
 + "parent chromosomes for crossover of genetic material.");
ftxtCrossover = new JFormattedTextField(
    NumberFormat.getNumberInstance());
l.setLabelFor(ftxtCrossover);

c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 1;
c.weightx = 0.0;

gb.setConstraints(l, c);
p.add(l);

ftxtCrossover.setEnabled(false);
ftxtCrossover.setFocusLostBehavior(
    JFormattedTextField.COMMIT_OR_REVERT);
ftxtCrossover.setValue(new Integer(problem.getCrossoverPoints()));
ftxtCrossover.addPropertyChangeListener(this);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 1;
c.weightx = 1.0;

gb.setConstraints(ftxtCrossover, c);
p.add(ftxtCrossover);

////////////////////////////////////////////////////////////////////////
// Mutation Rate
////////////////////////////////////////////////////////////////////////

l = new JLabel("Mutation Rate: ");
l.setToolTipText("The (random) likelihood for each bit to be flipped.");
ftxtMutation = new JFormattedTextField(
    NumberFormat.getNumberInstance());
l.setLabelFor(ftxtMutation);

c.anchor = GridBagConstraints.EAST;
c.fill = GridBagConstraints.NONE;
c.gridwidth = GridBagConstraints.RELATIVE;
c.gridy = 2;
c.weightx = 0.0;
gb.setConstraints(l, c);
p.add(l);

ftxtMutation.setEnabled(false);
ftxtMutation.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT);
ftxtMutation.setValue(new Double(problem.getMutationRate()));
ftxtMutation.addPropertyChangeListener(this);

C.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridy = 2;
c.weightx = 1.0;
gb.setConstraints(ftxtMutation, c);
p.add(ftxtMutation);

c = new GridBagConstraints();
c.anchor = GridBagConstraints.WEST;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 2;
c.gridy = 3;
c.weightx = 1.0;

////////////////////////////////////////////////////////////////////////
// Sift Mode
////////////////////////////////////////////////////////////////////////

chkSiftMode = new JCheckBox("Sift Terrain Tables");
chkSiftMode.setSelected(true);
chkSiftMode.setToolTipText("Rearranges the distribution of genetic parameters for the toroidal terrain tables to smoothly assign values across table edges.");
chkSiftMode.addItemListener(this);

gb.setConstraints(chkSiftMode, c);
p.add(chkSiftMode);

////////////////////////////////////////////////////////////////////////
subPanelB.add(p);

this.add(subPanelA);
this.add(subPanelB);
}

/**
 * Handles User Events from <code>JComboBox</code> objects.
 */
public void actionPerformed(ActionEvent e) {
    DecimalFormat f = new DecimalFormat();
    Object source = e.getSource();

    f.setDecimalFormatSymbols(new DecimalFormatSymbols(Locale.US));
f.setMinimumIntegerDigits(1);
f.setMaximumFractionDigits(4);
f.setMinimumFractionDigits(4);
f.setGroupingUsed(false);
    
    if (source == cmbFigure && problem.getMemeticMode()) {
    
*/
}
String name = (String) ((JComboBox) source).getSelectedItem();

if (name.equals("lena")) {
    problem.setImageToUtilize(name);
} else if (name.equals("logo")) {
    problem.setImageToUtilize(name);
} else if (name.equals("mandrill")) {
    problem.setImageToUtilize(name);
} else if (name.equals("peppers")) {
    problem.setImageToUtilize(name);
}

/*
 * Now with VisMTBMA, we need to enable several items on the screen
 * only appropriate for that problem. If it is selected, make those
 * changes and disable if any other algorithm is selected.
 * For future MA's, the name.equals could be ||'d or controlled in
 * other fashions.
 */
if (source == cmbProblem) {
    String name = (String) ((JComboBox) source).getSelectedItem();

    if (!name.equals("VisMTBMA")) {
        lblFigure.setEnabled(false);
        cmbFigure.setEnabled(false);
        chkMovementEnabled.setEnabled(false);
        ftxtMaxPerCell.setEnabled(false);
        ftxtFactor1.setEnabled(false);
        ftxtFactor2.setEnabled(false);
    }

    if (name.equals("Decep4")) {
        problem = new Decep4();
    } else if (name.equals("F1")) {
        problem = new F1();
    } else if (name.equals("F2")) {
        problem = new F2();
    } else if (name.equals("F3")) {
        problem = new F3();
    } else if (name.equals("F4")) {
        problem = new F4();
    } else if (name.equals("F5")) {
        problem = new F5();
    } else if (name.equals("Knap20")) {
        problem = new Knap20();
    } else if (name.equals("Knap80")) {
        problem = new Knap80();
    } else if (name.equals("Rastrigin")) {
        problem = new Rastrigin();
    } else if (name.equals("Schwefel")) {
        problem = new Schwefel();
    } else if (name.equals("Griewangk")) {
        problem = new Griewangk();
    } else if (name.equals("VisMTBMA")) {
        problem = new MTBMA();
        lblFigure.setEnabled(true);
        cmbFigure.setEnabled(true);
        chkMovementEnabled.setEnabled(true);
        ftxtMaxPerCell.setEnabled(true);
        ftxtFactor1.setEnabled(true);
        ftxtFactor2.setEnabled(true);
        problem.setImageToUtilize(figureList[0]);
    }

    // Notify vis config panel the problem has changed
visPanel.updateProblem(problem);

// Set UI Elements to their Default States
chkRandomSeed.setSelected(false);
chkSiftMode.setSelected(true);
chkTerrainMode.setSelected(false);
chkMovementEnabled.setSelected(false);

lblChromosome.setText("Chromosome Length: " + problem.getChromosomeLength());
lblTarget.setText("Target Value: " + f.format(problem.getTarget()));

ftxtCrossover.setValue(new Integer(problem.getCrossoverPoints()));
ftxtExperiments.setValue(new Integer(problem.getExperiments()));
ftxtGenerations.setValue(new Integer(problem.getGenerations()));
ftxtMutation.setValue(new Double(problem.getMutationRate()));
ftxtRandomSeed.setValue(new Long(problem.getSeed()));
ftxtSideLength.setValue(new Integer(problem.getSideLength()));
//Following two methods allow problem to override the text we display for the local search parameters.
ftxtFactor1.setValue(new Float(problem.getLowerScaleFactor()));
ftxtFactor2.setValue(new Float(problem.getUpperScaleFactor()));
txtaProbDesc.setText(problem.description());
lblFactor1.setText(problem.getFactor1());
lblFactor2.setText(problem.getFactor2());

/**
 * Returns a copy of the currently configured <code>Problem</code>.  
 * <p>
 * It is important that a copy is returned; otherwise the user could 
 * modify the <code>Problem</code> parameters during runtime 
 * (causing all sorts of havoc).
 * *
 * @return a clone of the current Problem
 */
public Problem getProblem() {
    return ((Problem) problem.clone());
}

/**
 * Handles User Events from <code>JCheckBox</code> objects.
 * *
 * @param source the source object of the event.
 */
public void itemStateChanged(ItemEvent e) {
    Object source = e.getItemSelectable();

    if (source == chkRandomSeed) {
        if (e.getStateChange() == ItemEvent.SELECTED) {  
            ftxtRandomSeed.setEnabled(true);
        } else {
            ftxtRandomSeed.setEnabled(false);
            problem.setSeed(System.currentTimeMillis());
            ftxtRandomSeed.setValue(
            new Long(problem.getSeed()));
        }
    }
}
new Long(problem.getSeed()));
}
else if (source == chkSiftMode) {
if (e.getStateChange() == ItemEvent.SELECTED) {
    problem.setSiftMode(true);
} else {
    problem.setSiftMode(false);
}
}
else if (source == chkTerrainMode) {
if (e.getStateChange() == ItemEvent.SELECTED) {
    problem.setTerrainMode(false);
    chkSiftMode.setEnabled(false);
    ftxtCrossover.setEnabled(true);
    ftxtMutation.setEnabled(true);
} else {
    problem.setTerrainMode(true);
    chkSiftMode.setEnabled(true);
    ftxtCrossover.setEnabled(false);
    ftxtMutation.setEnabled(false);
}
}
else if (source == chkMovementEnabled) {
    //MTBMA - controls whether or nor movement code is called
    //after evolution.
    if (e.getStateChange() == ItemEvent.SELECTED) {
        problem.setMovementEnabled(true);
    } else {
        problem.setMovementEnabled(false);
    }
}
}
/**
* Handles User Events from <code>JFormattedTextField</code> objects.
*/
public void propertyChange(PropertyChangeEvent e) {
    Object source = e.getSource();
    if (source == ftxtCrossover) {
        int value = ((Number) ftxtCrossover.getValue()).intValue();
        if (value < 0) {
            ftxtCrossover.setValue(new Integer(0));
            problem.setCrossoverPoints(0);
        } else {
            problem.setCrossoverPoints(value);
        }
    }
    else if (source == ftxtExperiments) {
        int value = ((Number) ftxtExperiments.getValue()).intValue();
        if (value < 1) {
            ftxtExperiments.setValue(new Integer(1));
            problem.setExperiments(1);
        } else {
            problem.setExperiments(value);
        }
    }
    else if (source == ftxtGenerations) {
        int value = ((Number) ftxtGenerations.getValue()).intValue();
        if (value < 1) {
            ftxtGenerations.setValue(new Integer(1));
            problem.setGenerations(1);
        } else {
            problem.setGenerations(value);
        }
    }
    else if (source == ftxtMutation) {
        double value = ((Number) ftxtMutation.getValue()).doubleValue();
        // MTBMA - controls whether or nor movement code is called
        // after evolution.
        if (e.getStateChange() == ItemEvent.SELECTED) {
            problem.setMovementEnabled(true);
        } else {
            problem.setMovementEnabled(false);
        }
    }
}

if (value < 0.0) {
    ftxtMutation.setValue(new Double(0.0));
    problem.setMutationRate(0.0);
} else {
    problem.setMutationRate(value);
}
} else if (source == ftxtRandomSeed) {
    long value = ((Number) ftxtRandomSeed.getValue()).longValue();
    problem.setSeed(value);
} else if (source == ftxtSideLength) {
    int value = ((Number) ftxtSideLength.getValue()).intValue();
    if (value < 3) {
        ftxtSideLength.setValue(new Integer(3));
        problem.setSideLength(3);
    } else {
        problem.setSideLength(value);
    }
}

} else if (source == ftxtFactor1) {
    //Keep the value within sane amounts as determined by MTBMA research
    float value = ((Number) ftxtFactor1.getValue()).floatValue();
    if (value >= problem.getUpperScaleFactor()) {
        ftxtFactor1.setValue(new Float(1.0));
        problem.setUpperScaleFactor(1.0f);
    } else if (value < 1) {
        ftxtFactor1.setValue(new Float(1.0));
        problem.setUpperScaleFactor(1.0f);
    } else {
        problem.setUpperScaleFactor(value);
    }
} else if (source == ftxtFactor2) {
    //Keep the value within sane amounts as determined by MTBMA research
    float value = ((Number) ftxtFactor2.getValue()).floatValue();
    if (value <= problem.getLowerScaleFactor()) {
        ftxtFactor2.setValue(new Float(1.4));
        problem.setLowerScaleFactor(1.4f);
    } else if (value > 2) {
        ftxtFactor2.setValue(new Float(1.4));
        problem.setLowerScaleFactor(1.4f);
    } else {
        problem.setLowerScaleFactor(value);
    }
} else if (source == ftxtMaxPerCell) {
    int value = ((Number) ftxtMaxPerCell.getValue()).intValue();
    if (value <= 1) {
        ftxtMaxPerCell.setValue(new Integer(1));
        problem.setMaxPerCell(1);
    } else if (value < problem.getSideLength() * problem.getSideLength()) {
        problem.setMaxPerCell(value);
    }
}
/**
 * Configures the panel and current <code>Problem</code> with the specified 
 * crossover and mutation rate parameters. Terrain mode is disabled.
 */
public void setTerrainParameters(int crossoverPoints, double mutationRate) {
    problem.setCrossoverPoints(crossoverPoints);
    problem.setMutationRate(mutationRate);
problem.setTerrainMode(false);
chkTerrainMode.setSelected(true);
ftxtCrossover.setEnabled(true);
ftxtCrossover.setValue(new Integer(crossoverPoints));
ftxtMutation.setEnabled(true);
ftxtMutation.setValue(new Double(mutationRate));
A.3 – Changes to Display.java

public class Display extends JPanel implements Runnable {

    private Analysis analysis;
    private BufferedImage img;
    private Point3D dataModel[];
    private Point3D dataModelCity[];
    private Point3D dataCopy[];
    private Problem problem;
    private Renderer renderMethod;
    private volatile Thread displayThread;
    // Control variable for MTBMA - determines if success or population is plotted
    private Boolean showCity;

    /**
     * Default Constructor
     */
    public Display() {
        analysis = null;
        problem = null;
        renderMethod = null;
        showCity = false;
        setBackground(Color.BLACK);
    }

    /**
     * Draws a double-buffered and rendered visualization to the screen.
     */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (displayThread == null) {
            return;
        }

        if (img == null || img.getWidth() != getWidth() || img.getHeight() != getHeight()) {
            img = (BufferedImage) createImage(getWidth(), getHeight());
        }

        if (img != null) {
            // Create a Deep Copy of the Data for the Renderer
            // Based on showCity, either copies the number of individuals per
            // location or the number of times an individual was most fit
            if (!showCity) {
                for (int i = 0, sz = dataModel.length; i < sz; i++) {
                    dataCopy[i] = (Point3D) dataModel[i].clone();
                }
            } else {
                for (int i = 0, sz = dataModelCity.length; i < sz; i++) {
                    dataCopy[i] = (Point3D) dataModelCity[i].clone();
                }
            }

            // Render Model and Draw Image to Screen
            renderMethod.render(dataCopy, img);
            g.drawImage(img, 0, 0, this);
        }
    }

    /**
     * Sets the internal representation of the number of individuals
     */
}
* per location in memetic algorithms
*/
public void setCity(int[] cityView) {
    for (int i = 0; i < cityView.length; ++i) {
        Point3D p = dataModelCity[i];
        p.setY(cityView[i] + 0.5);
    }
}

/**
 * Determines if the data passed to the renderer is the number of times
 * an individual was the most fit or if it is the number of individuals
 * at each location
 */
public void setShowCity(boolean show) {
    this.showCity = show;
}
A.4 – Changes to FlatRenderer.java

/**
 * Performs the labeling for the FlatRenderer, drawing labels for mutation
 * rate and crossover points axes.
 * @throws ArrayIndexOutOfBoundsException
 * @see Point3D
 */
private void doLabeling(Point3D model[], int sideLen, Graphics2D g2,
 int width, int height, int bw, int bh) {
    /* Calculate the x/y coordinates, as appropriate, for each mutation and
     * crossover point.  Note we only need to calculate one axis for either
     * crossovers OR mutations.
     */
    int mutPos[] = new int[sideLen];
    int crossPos[] = new int[sideLen];

    // Temporary variables.
    int x, y;
    // Crossover/mutation scratch variables.
    double xRate = 0;
    double mutRate = 0;
    // scale factor scratch variables for the memetic problems
    float scaleFact1 = 0.0f;
    float scaleFact2 = 0.0f;
    /* String format, so we don't cram too much on the screen at once.
     * Go to 4 decimals so that it doesn't look like there are 2 zeros
     * next to each other.
     */
    DecimalFormat df = new DecimalFormat("#####.####");
    int nudgeHeight = bh / 2;
    int nudgeWidth = bw / 2;

    for (int i = 0; i < sideLen; i++) {
        mutPos[i] = (int) (bh * (i + 1) - nudgeHeight);
        crossPos[i] = (int) (bw * (i + 1) - nudgeWidth);
    }

    // Mutation rate
    for (int i = 0; i < sideLen; i++) {
        if (!isShowScaleFactor() || !theProblem.getMemeticMode()) {
            // Get the mutation rate for this point.
            try {
                mutRate = theProblem.getMutationRate(i);
            } catch (java.lang.NullPointerException e) {
                return;
            }
        } else {
            // If user wants the local search parameters, display it instead
            scaleFact1 = theProblem.getScaleFactor1Points(i * sideLen);
        }

        // Calculate x and y.
        x = crossPos[sideLen - 1] + nudgeWidth + 3;
        y = mutPos[i];

        // Make tic marks cyan.
        g2.setColor(Color.CYAN);
        // Draw tic.
}
g2.drawLine(x, y, x + 5, y);

// Make text white.
g2.setColor(Color.WHITE);

// Actually draw mutation rate string.

// Scaling code
float scaleFactor = (float) (20.0 / sideLen);
if (scaleFactor > 1) {
    scaleFactor = 1;
} else if (scaleFactor < 0.75) {
    scaleFactor = (float) 0.75;
}
int fontSize = (int) (12 * scaleFactor);
// End scaling code

Font font = new Font("Monospaced", Font.BOLD, fontSize);
g2.setFont(font);
FontRenderContext frcMut = g2.getFontRenderContext();
LineMetrics lmMut;
if (!theProblem.getMemeticMode() || !isShowScaleFactor()) {
    lmMut = font.getLineMetrics(df.format(mutRate), frcMut);
} else if (isShowScaleFactor()) {
    lmMut = font.getLineMetrics(df.format(scaleFact1), frcMut);
} else {
    lmMut = null;
}

float stringHeight = lmMut.getAscent();
if (!theProblem.getMemeticMode() || !isShowScaleFactor()) {
    g2.drawString(df.format(mutRate), x + 6, y + (int) (stringHeight / 2));
} else {
    g2.drawString(df.format(scaleFact1), x + 6, y + (int) (stringHeight / 2));
}

// Crossover rate
for (int i = 0; i < sideLen; i++) {
    if (!isShowScaleFactor() || !theProblem.getMemeticMode()) {
        // Get crossover rate for this point.
        try {
            xRate = theProblem.getCrossoverPoints(i);
        } catch (java.lang.NullPointerException e) {
            return;
        }
    } else {
        // If the user wants the local search parameter, show it
        scaleFact2 = theProblem.getScaleFactor2Points(i);
    }

    // Calculate x and y.
    x = crossPos[i];
y = mutPos[sideLen - 1] + nudgeHeight + 1;

    // Make tic marks cyan.
g2.setColor(Color.CYAN);
    g2.drawLine(x, y, x, y + 8);
// Make text white.
g2.setColor(Color.WHITE);

// Actually draw crossover rate string.

// Scaling code.
float scaleFactor = (float) (20.0 / sideLen);
if (scaleFactor > 1) {
    scaleFactor = 1;
} else if (scaleFactor < 0.75) {
    scaleFactor = (float) 0.75;
}
int fontSize = (int) (12 * scaleFactor);

Font font = new Font("Monospaced", Font.BOLD, fontSize);
g2.setFont(font);

FontMetrics fMetXRate = g2.getFontMetrics();
int charWidth = fMetXRate.charWidth('0');

AffineTransform rotateXOver = AffineTransform.getRotateInstance(Math.PI / 3);
Font rotateXOverFont = font.deriveFont(rotateXOver);
g2.setFont(rotateXOverFont);

// Tweaked the y from 14 to 11.
if (!theProblem.getMemeticMode() || !isShowScaleFactor()) {
g2.drawString(df.format(xRate), Math.max(x - charWidth / 2, 0), y + 11);
} else if (isShowScaleFactor()) {
g2.drawString(df.format(scaleFact2), Math.max(x - charWidth / 2, 0), y + 11);
}
public void run() {
    Individual genBest;
    Thread thisThread = Thread.currentThread();

    long startTime;
    outer:
    while (curExp < maxExp) {
        // Update Status Information
        status.setExperiment(curExp + 1);
        analysis.resetGenerationCounter();

        while (curGen < maxGen) {
            // Break to Suspend / Terminate Thread
            if (geneticThread != thisThread) { break outer; }

            // Update Status Information
            status.setGeneration(curGen + 1);

            // Evolve Next Generation
            startTime = System.currentTimeMillis();
            pop.evolve();

            // Record Algorithm Runtime
            analysis.addRuntime(System.currentTimeMillis() - startTime);

            genBest = pop.getBestSolution();

            // Collect the current best for the best list.
            bestList[curExp][curGen] = genBest;

            if (expBest.getFitness() != genBest.getFitness()) {
                // Analyze New Data
                analysis.analyzeGeneration(genBest);

                // Update Display Information
                display.setData(genBest);
                display.setAnalysis(analysis);

                // Update Status Information
                status.setBestFitness(genBest.getFitness());

                expBest = genBest;

                //Problem involves local search parameters, for debug output
                //the params used for new most fit individual
                //if (probabilityMemeticMode) {
                // ParametersMTBMA tmp = problem.getParametersMTBMA().get(genBest.getIndex() / problem.getSideLength()).get(genBest.getIndex() % problem.getSideLength());
                // System.out.println("Generation best used scale factors " + tmp.scale_factor + "-" + tmp.scale_factor2);
                //}
            }
        }
    }
}
analysis.examineMovement(pop);
display.setCity(analysis.getCity());

if (runMax.getFitness() < genBest.getFitness()) {
    runMax = genBest;
}

// Toss the current best into the list of top
// solutions.
addToTopList(genBest);

// End Current Experiment if Target Solution Discovered
if (expBest.getFitness() == problem.getTarget()) {
    if (curGen + 1 < maxGen) {
        // Fill the remainder of the current experiment
        // with the final value, so the exported
        // time-convergence data will have a number
        // for every generation.
        for (int x = curGen + 1; x < maxGen; x++) {
            bestList[curExp][x] = expBest;
        }
        break;
    }
}

// Increment Current Generation
curGen++;

// Update Status Information
status.setAnalysis(analysis);
status.setProgress(curExp * maxGen + curGen);

if (stats != null && (curGen % 10 == 0)) {
    // Update the the convergence graph dimensions.
    stats.updateSpecs(maxGen, getMax().getFitness());
    // Once every ten generations update the display
    // of the top five solutions.
    if ((curGen % 10) == 0) {
        stats.updateTopFive(getTopList());
    }
}

// Update Analysis
analysis.analyzeExperiment(expBest);

// Update Status Information
status.setAnalysis(analysis);
status.setProgress((curExp + 1) * maxGen);

if (stats != null) {
    // Do a final update of the convergence graph
    // dimensions and top five list.
    stats.updateSpecs(maxGen, getMax().getFitness());
    stats.updateTopFive(getTopList());
}

/*
 * We re-initialize the experiment here to ensure a thread-safe
 * method, so the execution context is not reset across suspends
*/
* and resumes of this thread.
*/

// Create New Population
pop = new Population(problem);
// Make sure any local search parameters are properly distributed
pop.distribute_parameters(problem.getLowerScaleFactor(),
problem.getUpperScaleFactor());
expBest = pop.getBestSolution();

// Set Current Generation / Experiment
curGen = 0;
curExp++;
A.6 – Changes to HSRRenderer.java

/**
 * HSRPolygon shapes are rendered in order of largest distance from "camera" first (essentially over-painted),
 * smallest distance from "camera" last.
 * Grid axis labeling is also performed and when necessary the coordinates of a "point of interest" are visually represented.
 * To facilitate the rendering process, Matrix3D objects in the form of a Viewing Transformation Matrix,
 * a Model Transformation Matrix, and a Screen Transformation Matrix are used to obtain device coordinates from original hypothetical coordinates.
 *
 * @see HSRPoint3D
 * @see HSRPolygon
 * @see Matrix3D
 * @see Point3D
 * @see Renderer
 */

public void render(Point3D model[], BufferedImage canvas) {
    boolean scaling = false;
    int w = canvas.getWidth();
    int h = canvas.getHeight();
    int sideLen = theProblem.getSideLength();
    Point3D backupModel[] = new Point3D[sideLen * sideLen];
    /*make grounded ptOfInterest grounded at 0.0 and 'clone' of ptOfInterest*/

    if (isShowPeak()) {
        Point3D ptOfInterestPeak = getPeakPoint();
        ptOfPeakIntrstGnd = new Point3D(ptOfInterestPeak.getX(), 0.0,
                                        ptOfInterestPeak.getZ());
        ptOfPeakIntrstSubst = new Point3D(ptOfInterestPeak.getX(),
                                          ptOfInterestPeak.getY(),
                                          ptOfInterestPeak.getZ());
        posFactorPeak = sideLen / 2;
        interestIndexPeak = ((int) ptOfInterestPeak.getZ() + posFactorPeak) * sideLen + ((int) ptOfInterestPeak.getX() + posFactorPeak);
        interestColPeak = (int) ptOfPeakIntrstGnd.getX() + posFactorPeak;
        interestRowPeak = (int) ptOfPeakIntrstGnd.getZ() + posFactorPeak;
    }

    if (isShowWeightAvg()) {
        Point3D ptOfAvgInterest = getWeightAvgPoint();
        ptOfAvgIntrstGnd = new Point3D(ptOfAvgInterest.getX(), 0.0,
                                       ptOfAvgInterest.getZ());
        ptOfAvgIntrstSubst = new Point3D(ptOfAvgInterest.getX(),
                                         ptOfAvgInterest.getY(),
                                         ptOfAvgInterest.getZ());
        posFactorAvg = sideLen / 2;
        interestIndexAvg = ((int) ptOfAvgInterest.getZ() + posFactorAvg) * sideLen + ((int) ptOfAvgInterest.getX() + posFactorAvg);
        interestColAvg = (int) ptOfAvgIntrstGnd.getX() + posFactorAvg;
        interestRowAvg = (int) ptOfAvgIntrstGnd.getZ() + posFactorAvg;
    }
}
if (isAutoRotate()) {
    camOrt.setY(camOrt.getY() - 1);
}

double c = Math.cos(-Math.toRadians(camOrt.getY()));
double s = Math.sin(-Math.toRadians(camOrt.getY()));

MTM.identity();
MTM.set(0, 0, c); // Rotate Model
MTM.set(0, 2, -s);
MTM.set(2, 0, s);
MTM.set(2, 2, c);
MTM.set(1, 1, getScaleFactor());
MTM.postMultiply(VTM);

STM.set(0, 0, (double) (w / 2));
STM.set(1, 1, (double) -(h / 2));
STM.set(3, 0, (double) (w / 2));
STM.set(3, 1, (double) (h / 2));

/** some of these arrays need data filled in for a point of interest
   * regardless if labeling is needed or not */

establishLabelPoints(model, sideLen, xOverLabelPoints,
    xOverLabelPointAnchors,
    mutRateLabelPoints, mutRateLabelPointAnchors,
    xOverAngleOriginPoints,
    mutRateAngleOriginPoints, mainxOverLabelPoint,
    mainmutRateLabelPoint);

Graphics2D g2 = canvas.createGraphics();
g2.setBackground(new Color(255, 255, 125)); //old color Color.BLACK
g2.clearRect(0, 0, w, h);
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);

/** <code>HSRPoint3D</code> is a wrapper class for
   * <code>Point3D</code> objects and is used to organize
   * the array of <code>Point3D</code> objects passed to
   * the render method into <code>HSRPolygon</code> objects */

HSRPoint3D[] modelHSRs = new HSRPoint3D[model.length];

Vector finalPolygonSet = new Vector();

for (int row = 0; row < sideLen; row++) {
    for (int col = 0; col < sideLen; col++) {
        modelHSRs[row * sideLen + col] =
            new HSRPoint3D(model[row * sideLen + col], col,
                row);

        backupModel[row * sideLen + col] = new Point3D{
            model[row * sideLen + col].getX(),
            model[row * sideLen + col].getY(),
            model[row * sideLen + col].getZ();
        }
    }
}

altMTM.identity();
altMTM.set(0, 0, c);
altMTM.set(0, 2, -s);
altMTM.set(2, 0, s);
altMTM.set(2, 2, c);
altMTM.set(1, 1, getScaleFactor());
altMTM.postMultiply(altVTM);

computePolygons(modelHSRs, finalPolygonSet, sideLen);
for (int i = 0; i < finalPolygonSet.size(); i++) {
    HSRPolygon currentPolygon = (HSRPolygon) finalPolygonSet.elementAt(i);
    Point3D[] polygonPoints = currentPolygon.getAPoints();
    for (int j = 0; j < polygonPoints.length; j++) {
        polygonPoints[j] = ((Point3D) polygonPoints[j]).transform(altMTM);
    }
    currentPolygon.computeAvgZ((double) (sideLen - 1));
    currentPolygon.computeColorIntensity((double) (sideLen - 1));
}

//sort as third priority on the color intensity of the polygon rendered
sortOnColorIntensity(finalPolygonSet, new My_Integer(0), new My_Integer(finalPolygonSet.size() - 1));

//sort secondarily on distance from polygon to vertical center line
//of screen
sortOnCenterDist(finalPolygonSet, new My_Integer(0), new My_Integer(finalPolygonSet.size() - 1));

//sort primarily on z distance from polygon to screen plane
Collections.sort(finalPolygonSet);

outer:
for (int i = 0; i < finalPolygonSet.size(); i++) {
    HSRPolygon currentPolygon = (HSRPolygon) finalPolygonSet.elementAt(i);
    Point3D[] polygonPoints = currentPolygon.getBPoints();
    for (int j = 0; j < polygonPoints.length; j++) {
        Point3D bPoint = ((Point3D) polygonPoints[j]).transform(MTM);
        bPoint = bPoint.divide(bPoint.getW());
        polygonPoints[j] = bPoint.transform(STM);
        if (polygonPoints[j].getY() < 0.0
            || polygonPoints[j].getX() < 0.0
            || polygonPoints[j].getX() > (w - 1)) {
            scaling = true;
            break outer;
        }
    }
}

if (scaling) {
    scaleFactor = scaleFactor() * 0.9;
    //have to call render again in order to prevent long flicker
    //can't just let function go on after setting scaleFactor
    //because rendering goes haywire. Old model is compromised.
    //Therefore using a backup model.
    render(backupModel, canvas);
    return;
}

if (isShowLabel()) {
    drawLabeling(g2, sideLen, xOverLabelPoints, xOverLabelPointAnchors,
                 mutRateLabelPoints, mutRateLabelPointAnchors,
                 xOverAngleOriginPoints,
                 mutRateAngleOriginPoints, mainxOverLabelPoint,
                 mainmutRateLabelPoint);
}

//start: render the polygons
for (int j = 0; j < finalPolygonSet.size(); j++) {
    HSRPolygon renderedPoly = (HSRPolygon) finalPolygonSet.elementAt(j);
    Point3D[] thePoints = renderedPoly.getBPoints();
    int theSize = thePoints.length;
    int[] xCoords = new int[theSize];
    int[] yCoords = new int[theSize];
    for (int i = 0; i < theSize; i++) {
        xCoords[i] = Math.round((float) ((Point3D) thePoints[i]).getX());
        yCoords[i] = Math.round((float) ((Point3D) thePoints[i]).getY());
    }
    g2.setColor(new Color(0, (int) (255.0 * renderedPoly.getColorIntensity()), 0));
    g2.fillPolygon(xCoords, yCoords, theSize);
}

//end: render the polygons

/* draw the 'crosshairs' for the point of interest
 * fill an oval on point of interest
 * draw the dashed altitude guide for the point of interest */
if (isShowPeak()) {
    drawPointOfInterest(g2, ptOfPeakIntrstGnd, ptOfPeakIntrstSubst,
                        xOverAngleOriginPoints,
                        xOverLabelPointAnchors, mutRateAngleOriginPoints,
                        mutRateLabelPointAnchors, interestRowPeak, interestColPeak, Color.blue);
}
if (isShowWeightAvg()) {
    drawPointOfInterest(g2, ptOfAvgIntrstGnd, ptOfAvgIntrstSubst,
                        xOverAngleOriginPoints,
                        xOverLabelPointAnchors, mutRateAngleOriginPoints,
                        mutRateLabelPointAnchors, interestRowAvg, interestColAvg, Color.black);
}
g2.dispose();

/** This method achieves labeling of the grid axes of the
 * visualized terrain based grid onto the canvas.
 * @throws ArrayIndexOutOfBoundsException
 * @see Point3D */
private void drawLabeling(Graphics2D g2, int sideLen,
                          Point3D[] xOverLabelPoints, Point3D[] xOverLabelPointAnchors,
                          Point3D[] mutRateLabelPoints, Point3D[] mutRateLabelPointAnchors,
                          Point3D[] xOverAngleOriginPoints, Point3D[] mutRateAngleOriginPoints,
                          Point3D mainxOverLabelPoint, Point3D mainmutRateLabelPoint) {
    Font font = new Font("Monospaced", Font.BOLD, 10);
    //make compiler happy, "may not have been initialized" message
    float[] xOverTable = null;
double[] mutRateTable = null;
int maxXOverTxtLen = 0;
int maxMutRtTxtLen = 0;

DecimalFormat fmt = new DecimalFormat();
fmt.setDecimalFormatSymbols(new DecimalFormatSymbols(Locale.US));
fmt.setMinimumIntegerDigits(1);
fmt.setMaximumFractionDigits(4);
fmt.setMinimumFractionDigits(4);
fmt.setGroupingUsed(false);

xOverTable = new float[sideLen];
mutRateTable = new double[sideLen];

if (theProblem.isTerrainMode()) {
    for (int labelIndex = 0; labelIndex < sideLen; labelIndex++) {
        try {
            xOverTable[labelIndex] = 0;
            mutRateTable[labelIndex] = 0;
            if (theProblem.getMemeticMode() && isShowScaleFactor()) {
                // User asked for local search parameters instead of
                // mutation and crossover.
                xOverTable[labelIndex] = theProblem.getScaleFactor2Points(labelIndex);
                mutRateTable[labelIndex] = theProblem.getScaleFactor1Points(labelIndex *
                        theProblem.getSideLength());
            } else {
                xOverTable[labelIndex] = theProblem.getCrossoverPoints(labelIndex);
                mutRateTable[labelIndex] = theProblem.getMutationRate(labelIndex);
            }
        } catch (Exception e) {
            return;
        }
        String xOverTxtLenStr = Float.toString(xOverTable[labelIndex]);
        String mutRtTxtLenStr = "" + fmt.format(mutRateTable[labelIndex]);
        int tempMaxXOverTxtLen = xOverTxtLenStr.length();
        int tempMaxMutRtTxtLen = mutRtTxtLenStr.length();
        maxXOverTxtLen = (tempMaxXOverTxtLen > maxXOverTxtLen) ? tempMaxXOverTxtLen : maxXOverTxtLen;
        maxMutRtTxtLen = (tempMaxMutRtTxtLen > maxMutRtTxtLen) ? tempMaxMutRtTxtLen : maxMutRtTxtLen;
    }
} else {
    for (int labelIndex = 0; labelIndex < sideLen; labelIndex++) {
        xOverTable[labelIndex] = theProblem.getCrossoverPoints();
        mutRateTable[labelIndex] = theProblem.getMutationRate();
    }
    String xOverTxtLenStr = Integer.toString(theProblem.getCrossoverPoints());
    String mutRtTxtLenStr = "" + fmt.format(mutRateTable[0]);
    maxXOverTxtLen = xOverTxtLenStr.length();
    maxMutRtTxtLen = mutRtTxtLenStr.length();
}

String xOverSpacerStringMain = "";
int dashedInt = 8943;
char dashedChar = (char) dashedInt;
for (int appendIndA = 0; appendIndA < maxXOverTxtLen; appendIndA++) {
    if (appendIndA % 2 == 0) {
        xOverSpacerStringMain = xOverSpacerStringMain + dashedChar;
    }
}

xOverSpacerStringMain = xOverSpacerStringMain + dashedChar;

String mutRateSpacerStringMain = "";
for (int appendIndB = 0; appendIndB < maxMutRtTxtLen; appendIndB++) {
    if (appendIndB % 2 == 0) {
        mutRateSpacerStringMain = mutRateSpacerStringMain + dashedChar;
    }
}

mutRateSpacerStringMain = mutRateSpacerStringMain + dashedChar;

int lineExtension = 0;
for (int XOverLabels = 0; XOverLabels < sideLen; XOverLabels++) {
    Point3D xOverLabelPoint = xOverLabelPoints[XOverLabels];
    Point3D xOverAnchorPoint = xOverLabelPointAnchors[XOverLabels];
    double changeInYXOver = xOverLabelPoint.getY() - xOverAngleOriginPoints[XOverLabels].getY();
    double changeInXXOver = xOverLabelPoint.getX() - xOverAngleOriginPoints[XOverLabels].getX();

    AffineTransform rotateXOver = AffineTransform.getRotateInstance(Math.atan(changeInYXOver / changeInXXOver));
    // remember to adjust the rotate if the letters look upsidedown
    if (changeInXXOver < 0) {
        rotateXOver.rotate(Math.PI);
    }

    Font rotateXOverFont = font.deriveFont(rotateXOver);
    String XOverString = "" + xOverTable[XOverLabels];
    g2.setFont(rotateXOverFont);
    g2.setPaint(Color.black);
    String finalXOverStringMain = "";
    for (int appendXOverSpc = 0; appendXOverSpc < lineExtension; appendXOverSpc++) {
        finalXOverStringMain = finalXOverStringMain + xOverSpacerStringMain;
    }
    finalXOverStringMain = finalXOverStringMain + XOverString;
    g2.drawLine((int) xOverLabelPoint.getX(), (int) xOverLabelPoint.getY(), (int) xOverAnchorPoint.getX(), (int) xOverAnchorPoint.getY());
    g2.drawString(finalXOverStringMain, (int) xOverLabelPoint.getX(), (int) xOverLabelPoint.getY());
}

Point3D mutRateLabelPoint = mutRateLabelPoints[XOverLabels];
Point3D mutRateAnchorPoint = mutRateLabelPointAnchors[XOverLabels];
double changeInYMutRate = mutRateLabelPoint.getY() - mutRateAngleOriginPoints[XOverLabels].getY();
double changeInXMutRate = mutRateLabelPoint.getX() - mutRateAngleOriginPoints[XOverLabels].getX();

AffineTransform rotateMutRate = AffineTransform.getRotateInstance(}
Math.atan(changeInYMutRate / changeInXMutRate));

// remember to adjust the rotate if the letters look upsidedown
if (changeInXMutRate < 0) {
    rotateMutRate.rotate(Math.PI);
}

Font rotateMutRateFont = font.deriveFont(rotateMutRate);
String mutRateString = "" + frmt.format(mutRateTable[XOverLabels]);
g2.setFont(rotateMutRateFont);
//g2.setPaint(Color.yellow);
String finalmutRateStringMain = "";
for (int appendMutRateSpc = 0; appendMutRateSpc < lineExtension; appendMutRateSpc++) {
    finalmutRateStringMain = finalmutRateStringMain + mutRateSpacerStringMain;
}
finalmutRateStringMain = finalmutRateStringMain + mutRateString;
g2.drawLine((int) mutRateLabelPoint.getX(),
    (int) mutRateLabelPoint.getY(), (int) mutRateAnchorPoint.getX(),
    (int) mutRateAnchorPoint.getY());
g2.drawString(finalmutRateStringMain,
    (int) mutRateLabelPoint.getX(), (int) mutRateLabelPoint.getY());
lineExtension = ++lineExtension % 3;

Font fontMain = new Font("Monospaced", Font.BOLD, 10);
g2.setFont(fontMain);
g2.setPaint(Color.black);

String mnXOverString = "X over Pts";
if (theProblem.getMemeticMode() && isShowScaleFactor()) {
    //Change the label to what the problem specifies.
    mnXOverString = theProblem.getFactor1();
}
g2.drawString(mnXOverString, (int) (xOverAngleOriginPoints[sideLen / 2].getX() - 0.25 * (xOverLabelPointAnchors[sideLen / 2].getX() - xOverAngleOriginPoints[sideLen / 2].getX())),
    (int) (xOverAngleOriginPoints[sideLen / 2].getY() - 0.25 * (xOverLabelPointAnchors[sideLen / 2].getY() - xOverAngleOriginPoints[sideLen / 2].getY())));

String mmmutRateString = "Mutation Rt.";
if (theProblem.getMemeticMode() && isShowScaleFactor()) {
    //Change the label to what the problem specifies.
    mmmutRateString = theProblem.getFactor2();
}
g2.drawString(mmmutRateString, (int) (mutRateAngleOriginPoints[sideLen / 2].getX() - 0.25 * (mutRateLabelPointAnchors[sideLen / 2].getX() - mutRateAngleOriginPoints[sideLen / 2].getX())),
    (int) (mutRateAngleOriginPoints[sideLen / 2].getY() - 0.25 * (mutRateLabelPointAnchors[sideLen / 2].getY() - mutRateAngleOriginPoints[sideLen / 2].getY())));
public class Individual implements Cloneable {

    private char code[];
    private double fitness;
    private int index;
    private Problem problem;
    /**
     * This is the mean standard error
     */
    private float mse = 0.0f;
    /**
     * This is the signal to noise ratio
     */
    private double psnr = 0.0;
    /**
     * Genotype is a set of codevectors (Codebook)
     */
    private ArrayList<ArrayList<Float>> genotype = new ArrayList<ArrayList<Float>>();
    /**
     * This is the partition encoded by the codevectors present in the genotype
     */
    Map<Integer, ArrayList<Integer>> phenotype = new HashMap();
    /**
     * Individual's position in the terrain
     */
    private Pair<Integer, Integer> position;
    /**
     * This is the quantization error
     */
    private double q_err = 0.0;
    /**
     * Individual's rank in the population
     */
    private int rank;

    /**
     * Creates an <code>Individual</code> with a randomly generated chromosome
     * of the specified <code>Problem</code> chromosome length and index.
     */
    public Individual(Problem problem, int index) {
        int length = problem.getChromosomeLength();
        this.problem = problem;
        this.index = index;
        code = new char[length];
        for (int i = 0; i < length; i++) {
            code[i] = ((problem.nextInt(2) == 1) ? '1' : '0');
        }
        this.setPosition();
    }

    /**
     * Creates an <code>Individual</code> whose chromosome is identical to
     * the given character array.
     */
    public Individual(char code[]) {
        this.code = new char[code.length];
        System.arraycopy(code, 0, this.code, 0, code.length);
        problem = null;
    }
}
/**
 * Converts an unsigned binary integer to binary reflected Gray code.
 */
public static int binaryToGray(int bin) {
    return bin ^ (bin >>> 1);
}

/**
 * Returns a new <code>Individual</code> that is a deep copy of this
 * <code>Individual</code>.
 */
public Object clone() {
    Individual c = null;
    try {
        c = (Individual) super.clone();
        c.setCode(code);
        c.setFitness(fitness);
        c.setIndex(index);
        c.setProblem(problem);
        // Make sure to clone MTBMA details for the Individual
        c.setmse(mse);
        c.setpsnr(psnr);
        c.setGenotype(genotype);
        c.setPhenotype(phenotype);
        c.setPosition(position);
        c.setRank(rank);
        c.setQ_ERR(q_err);
    } catch (CloneNotSupportedException e) {
        System.err.println(e.getMessage());
    }
    return c;
}

/**
 * Method creates a new individual given the genotype, position and rank.
 * Depends on constructor to set default values for other member variables.
 *
 * @param genotype is the new codebook for the individual
 * @param x is the x position in the terrain to aid in finding neighbors
 * @param y is the y position in the terrain to aid in finding neighbors
 * @param rank is the individuals rank in the population
 *
 * @return new individual
 */
public Individual create_individual(ArrayList<ArrayList<Float>> genotype, Integer x,
        Integer y, Integer rank) {
    Individual ind = new Individual();
    // Make sure to initialize the new mtbma values
    ind.genotype = genotype;
    ind.setPosition(new Pair<Integer, Integer>(x, y));
    ind.setRank(rank);
    return ind;
}

/**
 * Returns the individual's genotype
 */
public ArrayList<ArrayList<Float>> getGenotype() {
    ArrayList<ArrayList<Float>> returnGenotype = new ArrayList<ArrayList<Float>>(this.genotype);
    return returnGenotype;
public Float getMSE() {
    return this.mse;
}

/**
 * Returns the current Individual's position
 */
public Pair<Integer, Integer> getPosition() {
    return position;
}

/**
 * Returns the individual's genotype
 */
public Map<Integer, ArrayList<Integer>> getPhenotype() {
    Map<Integer, ArrayList<Integer>> returnValue = new HashMap<Integer, ArrayList<Integer>>(this.phenotype);
    return returnValue;
}

/**
 * Method to retrieve the Individual's Peak Signal to Noise Ratio
 */
public Double getPSNR() {
    return this.psnr;
}

/**
 * Method to retrieve the Individual's q_err
 */
public Double getQ_Err() {
    return q_err;
}

/**
 * Sets the genotype for an Individual or loads the codebook to utilize when evaluating fitness of the Individual.
 */
public void setGenotype(ArrayList<ArrayList<Float>> newGenotype) {
    this.genotype = newGenotype;
}

/**
 * Update an individual's mean standard error
 */
public void setMSE(Float newMSE) {
    this.mse = newMSE;
}

/**
 * Sets the individual's phenotype for deep copies
 */
public void setPhenotype(Map<Integer, ArrayList<Integer>> phen) {
    this.phenotype = phen;
}

public void setPosition() {
    Pair<Integer, Integer> pos = new Pair<Integer, Integer>(
        this.index % problem.getSideLength(),
        (this.index / problem.getSideLength()) % problem.getSideLength());
    this.position = pos;
}
/**
* Set the individual's peak signal to noise ratio
* Measures the 'correctness' of the VQ'd image versus the original
* Higher is a better representation of the original
*/
public void setpsnr(Double newPSNR) {
    this.psnr = newPSNR;
}

/**
* Set the Individual's q_err value
* Measure of quantization error in the VQ process
*/
public void setQ_ERR(double newQ_Err) {
    this.q_err = newQ_Err;
}
/* MTBMA.java
 * Terrain-Based Genetic Algorithm Visualization Project
 * Copyright 2011 CSUS. All rights reserved.
 * Use is subject to license terms.
 */

//import java.util.*;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
/**
 * Implementation of Azevedo's mTBMA algorithm.
 */
public class MTBMA extends Problem {
    //Set default values for internal variables
    private String imageToUse = "lena";
    private int N = 256;
    private int K = 3;
    private double dinf = 0;
    private double dsup = 255;
    private ArrayList<ArrayList<Float>> X = new ArrayList<ArrayList<Float>>();
    private VQ vectorQ = new VQ();
    //Note the next two values are pulled by display and configuration classes
    //Sets the name for the local search parameters
    private String Factor1 = "Lower Bound: ";
    private String Factor2 = "Upper Bound: ";
    ArrayList<ArrayList<ParametersMTBMA>> parameters;

    public MTBMA() {
        // Default Values
        setChromosomeLength(30);
        setCrossoverPoints(2);
        setExperiments(3);
        setGenerations(2);
        setMutationRate(0.05);
        setSideLength(5);
        setSiftMode(false);
        setTarget(1.0);
        setTerrainMode(true);
        setMemeticMode(true);
        N = 256;
        K = 3;
        dinf = 0;
        dsup = 255;
        vectorQ.N = N;
        vectorQ.K = K;
        vectorQ.dinf = dinf;
        vectorQ.dsup = dsup;
        X = load_signal(imageToUse, vectorQ);
        vectorQ.X = X;
        parameters = new ArrayList<ArrayList<ParametersMTBMA>>() {
        }
    }

    /**
    * Implement problem's description
    * @return String with textual description of the problem solved
    */
public String description() {
    return new String("Implementation of Azevedo's Motioner Terrain Based Memetic Algorithm.");
}

/**
 * Implements problem's evaluate function. Responsible for determining the
 * fitness of an individual by determining the PSNR for the individual.
 * @param solution an individual to evaluate
 * @return a number representing the fitness of the solution
 */
public double evaluate(Individual solution) {
    solution.setFitness(evaluateIndividual(solution, vectorQ));
    // Debug output - left in to help pace the system.
    // This method is somewhat slow and without the feedback it appears
    // the system is stalled.
    System.out.println("Fitness for index " + solution.getIndex() + " was " + solution.getFitness());
    return solution.getFitness();
}

/**
 * Evaluates the fitness of a given individual
 * @param I the individual to evaluate
 * @param myVQ the VQ of the mTBMA
 * @return the fitness value for a given individual
 */
double evaluateIndividual(Individual I, VQ myVQ) {
    Double qerr = I.getQ_Err();
    float qerrFloat = qerr.floatValue();
    I.setmse(myVQ.MSE(qerrFloat));
    I.setpsnr((double) myVQ.PSNR(I.getmse()));
    return (1 / (1 + I.getmse()));
}

/**
 * Method to return the local search parameter description
 * @return string to use in configpanel and display output
 */
public String getFactor1() {
    return this.Factor1;
}

/**
 * Method to return the local search parameter description
 * @return string to use in configpanel and display output
 */
public String getFactor2() {
    return this.Factor2;
}

/**
 * Method to return the image the user has indicated we should utilize
 * @return string of name of image
 */
public String getImageToUtilize() {
    return this.imageToUse;
}

/**
 * Method to return the distributed parameters used in the execution of
 * mTBMA
 * @return list of parameters
 */
public ArrayList< ArrayList< ParametersMTBMA>> getParametersMTBMA() {
    return this.parameters;
}
/**
 * Method to return the VQ for the particular instance
 * Used in mutation and crossover
 */
public VQ getVQ() {
    return vectorQ;
}
/**
 * Implements the Problem method to determine if the problem maintains
 * historical data.
 * @return false
 */
public boolean isHistorical(Individual bestSolution) {
    return false;
}
/**
 * Method to seed the individual's codebook
 * @param id The problem to solve
 * @param myMTBMA The MTBMA to receive the population
 * @param myVQ The VQ that will be used to solve the population
 * @return
 */
public Individual loadCodebook(String id, int index, VQ myVQ) {
    index += 1;
    String path = id + "/" + id + ".dic." + myVQ.N + "." + index + ".dat";
    //System.out.println("Path for pop is " + path);
    ArrayList<ArrayList<Float>> W = new ArrayList<ArrayList<Float>>();
    ArrayList<Float> w = new ArrayList<Float>();
    int j = 1;
    //Reads in a codebook as formatted by Azevedo's mTBMA implementation
    //Assumes all files are in <location of jar>/<image name>
    try {
        File file = new File(path);
        Scanner input = new Scanner(file);
        while (input.hasNext()) {
            String nextToken = input.next();
            float x = Float.parseFloat(nextToken);
            w.add(x);
            if (j % myVQ.K == 0) {
                ArrayList<Float> tmp = new ArrayList<Float>();
                //tmp.setSize(w.size());
                //Collections.copy(tmp, w);
                tmp.ensureCapacity(w.size());
                tmp.addAll(w);
                W.add(tmp);
                w.clear();
            }
            ++j;
        }
        input.close();
    } catch (FileNotFoundException e) {
        System.err.println("Error when opening " + path + ".");
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println("Exception while reading the file" + ioe);
    }
}
Individual individual = new Individual().create_individual(W, 0, 0, 0);
return individual;

/**
 * Method to load the image into the VQ
 * @param id Identifier of the codebook
 * @param myVQ The VQ to receive the codebook
 * @return
 */
public static ArrayList<ArrayList<Float>> load_signal(String id, VQ myVQ) {
    String path = id + "/" + id + ".dat";
    ArrayList<ArrayList<Float>> X = new ArrayList<ArrayList<Float>>();
    ArrayList<Float> w = new ArrayList<Float>();
    int j = 1;
    try {
        FileInputStream fstream = new FileInputStream(path);
        DataInputStream in = new DataInputStream(fstream);
        while (in.available() != 0) {
            String input = in.readLine();
            float x = Float.parseFloat(input);
            w.add(x);
            if (j % myVQ.K == 0) {
                ArrayList<Float> tmp = new ArrayList<Float>();
                tmp.ensureCapacity(w.size());
                tmp.addAll(w);
                X.add(tmp);

                //System.out.println( "X -- " + X.firstElement().firstElement() + "---" + X.get(0).get(0) + " --- " + x );
                //System.out.println( X );
                w.clear();
            }
            ++j;
        }
    } catch (FileNotFoundException e) {
        System.err.println("Error when opening " + path + ".");
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println("Exception while reading the file" + ioe);
    }
    return X;
}

/**
 * Method to set the image the user indicates we should use
 */
public void setImageToUtilize(String imageName) {
    this.imageToUse = imageName;
    X = load_signal(imageToUse, vectorQ);
    vectorQ.X = X;
}

/**
 * Method to set our instance of the parameters to what the user has specified
 * and distributed as the user saw fit
 */
public void setParametersMTBMA(ArrayList<ArrayList<ParametersMTBMA>> params) {
    this.parameters.clear();
    //this.parameters.setSize(params.capacity());
    this.parameters.ensureCapacity(params.size());
    //Collections.copy(this.parameters, params);
    for (int i = 0; i < params.size(); ++i) {
        this.parameters.add(i, params.get(i));
    }
    if (isSiftMode()) {
        this.parameters = sift(this.parameters);
    }
}

public void setX(ArrayList<ArrayList<Float>> newX) {
    this.X = newX;
    vectorQ.X = this.X;
}

/**
 * Method to distribute the parameters across the terrain as VisTBGA sifts parameters
 */
private ArrayList<ArrayList<ParametersMTBMA>> sift(ArrayList<ArrayList<ParametersMTBMA>> params) {
    ArrayList<ArrayList<ParametersMTBMA>> newParams = new ArrayList<ArrayList<ParametersMTBMA>>(params);
    float scale1[];
    float scale2[];
    int counter = 0;
    scale1 = new float[params.size() * params.size()];
    scale2 = new float[scale1.length];
    for (int i = 0; i < getSideLength(); i++) {
        ArrayList<ParametersMTBMA> innerArrayList = new ArrayList<ParametersMTBMA>(params.get(i));
        for (int j = 0; j < innerArrayList.size(); j++) {
            scale1[counter] = innerArrayList.get(j).scale_factor;
            scale2[counter++] = innerArrayList.get(j).scale_factor;
        }
    }
    /*  for (int z = 0; z < scale1.length; ++z)
        System.out.print(scale1[z] + " ");
    System.out.println();
    */
    boolean siftFlag = false;
    int tableBottom;
    int tableTop;
    float siftedScale1[];
    float siftedScale2[];
    // Allocate Space for Sifted Tables
    siftedScale1 = new float[scale1.length];
    siftedScale2 = new float[scale2.length];
    tableBottom = 0;
    tableTop = siftedScale1.length - 1;
    for (int i = 0; i < siftedScale1.length; i++) {
if (siftFlag) {
    // Sifts to Bottom
    siftedScale1[tableBottom] = scale1[i];
    siftedScale2[tableBottom] = scale2[i];
    tableBottom++;
} else {
    // Sifts to Top
    siftedScale1[tableTop] = scale1[i];
    siftedScale2[tableTop] = scale2[i];
    tableTop--;
}
siftFlag = !siftFlag;
}
counter = 0;
/* for( int z = 0; z < scale1.length; ++z)
 { System.out.print(siftedScale1[z] + " ");
 }*/
for (int i = 0; i < getSideLength(); i++) {
    ArrayList<ParametersMTBMA> innerVector = new ArrayList<ParametersMTBMA>(params.get(i));
    for (int j = 0; j < innerVector.size(); j++) {
        ParametersMTBMA scales = new ParametersMTBMA();
        scales.scale_factor = siftedScale1[counter];
        scales.scale_factor2 = siftedScale2[counter++];
        innerVector.set(j, scales);
    }
    newParams.set(i, innerVector);
}
return newParams;
} */
/**
 * @author brad
 */
public class Pair<A, B> {
    /**
     * fst is the internal representation of the first object in the pair
     */
    private A fst;

    /**
     * snd is the internal representation of the second object in the pair
     */
    private B snd;

    /**
     * Constructor with a given set of objects for the pair
     */
    public Pair(A fst, B snd) {
        this.fst = fst;
    }
}
this.snd = snd;

/**
 * Method to retrieve the internally stored first object
 * @return the first object of the pair
 */
public A getFirst() {
    return fst;
}

/**
 * Method to retrieve the internally stored second object
 * @return the second object of the pair
 */
public B getSecond() {
    return snd;
}

/**
 * Method to set the internally stored first object
 */
public void setFirst(A v) {
    fst = v;
}

/**
 * Method to set the internally stored second object
 */
public void setSecond(B v) {
    snd = v;
}

/**
 * Method to return the string representation of the pair objects
 * Assumes objects stored within fst/snd can be output via std calls
 * Assumes fst and snd are valid and not null
 * @return string "Pair[ fst , snd ]"
 */
@Override
public String toString() {
    return "Pair[" + fst + "," + snd + "]";
}

/**
 * Method to determine if objects are equal
 * Uses override function for Object.equals
 * @param x is the first object to compare
 * @param y is the second object to compare
 * @return true if x & y exist and x.equals( y ) returns true
 */
private static boolean equals(Object x, Object y) {
    return (x == null && y == null) || (x != null && x.equals(y));
}

/**
 * Method overrides Object.equals to confirm if two objects in the pair
 * equal each other. Finds fst/snd for both objects and compares each
 * part separately.
 */
/**
 * @param other is the second object to compare
 * @return true if all objects equal each other
 */
@Override
public boolean equals(Object other) {
    return other instanceof Pair
        && equals(fst, ((Pair) other).fst)
        && equals(snd, ((Pair) other).snd);
}

/**
 * Method computes a hash code for the pair to avoid collisions
 * Simple hash function used for this implementation.
 * Could lead to collisions if objects selected are not carefully crafted.
 * @return hashcode calculated based off of pair members
 */
@Override
public int hashCode() {
    if (fst == null) {
        return (snd == null) ? 0 : snd.hashCode() + 1;
    } else if (snd == null) {
        return fst.hashCode() + 2;
    } else {
        return fst.hashCode() * 17 + snd.hashCode();
    }
}

public static <A, B> Pair<A, B> of(A a, B b) {
    return new Pair<A, B>(a, b);
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

public class ParametersMTBMA {
    /**
     * Scale factor for Lee et al. update equation.
     */
    public Float scale_factor = new Float(0.0f);

    /**
     * Number of learning iterations
     */
    public Float scale_factor2 = new Float(0.0f);

    /**
     * Constructor method with given factors
     *
     * @param factor1 is the scale factor for Lee update equations
     * @param factor2 is the number of learning iterations
     */
    public ParametersMTBMA(float factor1, float factor2) {
        this.scale_factor = factor1;
        this.scale_factor2 = factor2;
    }
public ParametersMTBMA() {
    this.scale_factor = 0.0f;
    this.scale_factor2 = 0.0f;
}
A.9 – Changes to Population.java

```java
public class Population {
    private Individual genBest;
    private Individual solutions[];
    private Individual newSolutions[];

    private Problem problem;
    private RandomMTBMA myRandom = new RandomMTBMA();

    private ArrayList<ArrayList<Integer>> city_ok = new ArrayList<ArrayList<Integer>>();
    private ArrayList<ArrayList<Integer>> qtd;

    private ArrayList<Pair<Integer, Integer>> positions;

    /**
     * Generates a new <code>Population</code> of <code>Individual</code>
     * solutions to solve a particular <code>Problem</code>. The size of the
     * population and whether or not terrain tables are built depend on
     * how the <code>Problem</code> is configured.
     */

    public Population(Problem problem) {
        this.problem = problem;

        // Build Terrain Tables
        if (problem.isTerrainMode())
            problem.buildTerrain();

        // Allocate Space for Populations
        solutions = new Individual[problem.getPopulationSize()];
        newSolutions = new Individual[problem.getPopulationSize()];

        // Initialize the terrain and the vectors of the terrain
        city_ok.setSize(solutions.length);
        city_ok.ensureCapacity(solutions.length);
        for (int i = 0; i < solutions.length; ++i) {
            city_ok.add(i, new ArrayList<Integer>(solutions.length));
            city_ok.get(i).setSize(solutions.length);
            city_ok.get(i).ensureCapacity(solutions.length);
            for (int j = 0; j < solutions.length; ++j) {
                // used to store the index (ID) of the city
                // which is located at (i,j) coordinates of the
                // terrain
                city_ok.get(i).add(j, 0);
            }
        }

        if (problem.getMemeticMode())
        {
            // Initialize qtd
            qtd = new ArrayList<ArrayList<Integer>>();
            qtd.setSize(problem.getSideLength());
            qtd.ensureCapacity(problem.getSideLength());
            for (int i = 0; i < problem.getSideLength(); ++i) {
                qtd.add(i, new ArrayList<Integer>(problem.getSideLength()));
                qtd.get(i).setSize(problem.getSideLength());
                qtd.get(i).ensureCapacity(problem.getSideLength());
                for (int j = 0; j < problem.getSideLength(); ++j) {
                    //
                }
            }
        }
    }
}
```
```java
qtd.get(i).add(j, 1);
}
}

// Initialize positions
positions = new ArrayList<Pair<Integer, Integer>>();
// positions.setSize(solutions.length);
positions.ensureCapacity(solutions.length);
// Setup the vector - allocate the memory
for (int i = problem.getSideLength() - 1; i >= 0; --i) {
    for (int j = problem.getSideLength() - 1; j >= 0; --j) {
        positions.add(new Pair<Integer, Integer>(i, j));
    }
}

for (int i = 0; i < solutions.length; i++) {
    // Generate New Individual with Specified Length and Index
    solutions[i] = new Individual(problem, i);
    if (problem.getMemeticMode()) {
        solutions[i] = loadCodebook(solutions[i], problem.getImageToUtilize(), i, problem.getVQ());
        create_phenotype(solutions[i], problem.getVQ());
    }
    // Evaluate Individual's Fitness
    double fitness = problem.evaluate(solutions[i]);
    solutions[i].setFitness(fitness);
    // Determine Best Solution
    if (i == 0)
        genBest = solutions[i];
    else
        genBest = problem.compare(genBest, solutions[i]);
}

/**
 * Creates individual's phenotype.
 * The phenotype of the individual (i.e., partition of the input space
 * encoded in the VQ codebook) is calculated to assess the quantization
 * error.
 * @param I A given individual.
 */
private void create_phenotype(Individual I, VQ myVQ) {
    Individual newIndividual = new Individual();
    newIndividual = myVQ.calc_partition(I, myVQ.X);
    I = newIndividual;
}

/**
 * Method to perform crossover of two individuals
 * @param I1 is parent one
 * @param I2 is parent two
 * @return the resulting individual
 */
Individual crossover(Individual I1, Individual I2, VQ myVQ) {
    ArrayList<ArrayList<Float>> genotype1 = I1.getGenotype();
    ArrayList<ArrayList<Float>> genotype2 = I2.getGenotype();
```
int N = (int) myRandom.random_range(myRandom.getSeed(), 1, myVQ.N);

if (N > myVQ.N) {
    System.out.println("Error: # of Xover points cannot be higher than the genotype's size.");
    System.exit(0);
}

ArrayList<Integer> numbers = new ArrayList<Integer>();
for (int i = 0; i < myVQ.N; ++i) {
    numbers.add(i);
}

ArrayList<Integer> locus_pts = new ArrayList<Integer>();
if (N < myVQ.N) {
    for (int i = 0; i < N; ++i) {
        int j = (int) myRandom.random_range(myRandom.getSeed(), 0, numbers.size());
        int locus = numbers.get(j);
        locus_pts.add(locus);
        numbers.remove(j);
    }
} else {
    locus_pts = numbers;
}

Collections.sort(locus_pts);

int nbblocks = (N == myVQ.N) ? N : N + 1;
int inf = 0;
int c1 = 0;
int c2 = 0;
for (int i = 0; i < nbblocks; ++i) {
    float u = myRandom.value_range(myRandom.getSeed(), 0, 1);
    int l = 0;
    if (i == nbblocks - 1) {
        l = myVQ.N;
    } else {
        l = locus_pts.get(i);
    }
    if (u < 0.5) {
        for (int j = inf; j < l; ++j) {
            genotype1.set(j, I1.getGenotype().get(j));
            genotype2.set(j, I2.getGenotype().get(j));
        }
        c1 = c1 + (l - inf); /* Count # of inherited genes from I1 by gen1 */
    } else {
        for (int j = inf; j < l; ++j) {
            genotype1.set(j, I1.getGenotype().get(j));
            genotype2.set(j, I2.getGenotype().get(j));
        }
        c2 = c2 + (l - inf); /* Count # of inherited genes from I2 by gen1 */
    }
    if (i != nbblocks - 1) {
        inf = locus_pts.get(i);
    }
}

/* Return the offspring which has the largest part of the
 * parent from which it will inherit its position.
 * This aims at mantaining diversity */
if (c1 >= c2) {
Individual offspring1 = new Individual().create_individual(genotype1, I1.getPosition().getFirst(), I1.getPosition().getSecond(), I1.getRank());
offspring1.setProblem(I1.getProblem());
offspring1.setPosition(I1.getPosition());
offspring1.setCode(I1.getCode());
offspring1.setIndex(I1.getIndex());
return offspring1;
}

Individual offspring2 = new Individual().create_individual(genotype2, I1.getPosition().getFirst(), I1.getPosition().getSecond(), I1.getRank());
offspring2.setProblem(I1.getProblem());
offspring2.setPosition(I1.getPosition());
offspring2.setCode(I1.getCode());
offspring2.setIndex(I1.getIndex());
return offspring2;
}

/**
 * Method to perform crossover between individuals
 * @param I1 First individual
 * @param I2 Second individual
 * @param looser tracks which individual 'wins' more
 * @return The resulting individual
 */
private Pair< Individual, Integer > crossover(Individual I1, Individual I2, Integer looser, VQ myVQ ) /* N-Point Xover operator */ {
try
{
if( myRandom.getSeed() == 0 )
{
    myRandom.setSeed( System.currentTimeMillis() );
}
}
catch( Exception e )
{
}

Individual ind1 = (Individual) I1;
Individual ind2 = (Individual) I2;

ArrayList<ArrayList<Float>> genotype1 = ind1.getGenotype();
ArrayList<ArrayList<Float>> genotype2 = ind2.getGenotype();
int N = new Integer(0);
double myValue = new Double(0.0);
myValue = myRandom.random_range(myRandom.getSeed(), 1, myVQ.N);
N = (int) myValue;

if (N > myVQ.N) {
    System.err.println("Error: # of Xover points cannot be higher than the genotype's size.");
    System.exit(0);
}

ArrayList<Integer> numbers = new ArrayList<Integer>();
for (int i = 0; i < myVQ.N; ++i) {
    numbers.add(i);
}
Collections.shuffle(numbers);

ArrayList<Integer> locus_pts = new ArrayList<Integer>();
if (N < myVQ.N) {
    for (int i = 0; i < N; ++i) {

locus_pts.add(numbers.get(i));
}
}

Collections.sort(locus_pts);

int nblocks = (N == myVQ.N) ? N : N + 1;
int inf = 0;

//cout << N << "-point crossover...
"

int c1 = new Integer(0);
int c2 = new Integer(0);
for (int i = 0; i < nblocks; ++i) {
double u = new Double(0.0);
u = myRandom.value_range(myRandom.getSeed(), 0.0f, 1.0f);

int l = 0;
if (i == nblocks - 1) {
l = myVQ.N;
} else {
l = locus_pts.get(i);
}

//cout << "l(" << l << ", inf(" << inf << ", l - inf(" << l - inf << ")\n"
//cout << "c(" << c1 << ") << c2 << ")\n"

if (u < 0.5) {
for (int j = inf; j < l; ++j) {
genotype1.set(j, ind1.getGenotype().get(j));
genotype2.set(j, ind2.getGenotype().get(j));
} c1 = c1 + (l - inf); /* Count # of inherited genes from I1 by gen1 */
} else {
for (int j = inf; j < l; ++j) {
genotype1.set(j, ind2.getGenotype().get(j));
genotype2.set(j, ind1.getGenotype().get(j));
} c2 = c2 + (l - inf); /* Count # of inherited genes from I1 by gen1 */
}

if (i != nblocks - 1) {
inf = locus_pts.get(i);
}

/* Return the offspring which has the largest part of the
* parent from which it will inherit its position.
* This aims at maintaining diversity */
//cout << "c(" << c1 << ", " << c2 << ")\n"
if (c1 >= c2) {
Individual offspring1 = ind1;
offspring1.setGenotype(genotype1);
offspring1.setPosition(ind1.getPosition());
offspring1.setRank(ind1.getRank());
looser = 1;
//cout << "rate: " << 1 - c2 / (c1 + c2 + .0);
Pair< Individual, Integer > result = new Pair( offspring1, looser );
return result;
}

Individual offspring2 = ind2;
offspring2.setGenotype(genotype2);
offspring2.setPosition(ind1.getPosition());
offspring2.setRank(ind1.getRank());
looser = 2;
//cout << "rate: " << 1 - c1 / (c1 + c2 + .0);
Pair< Individual, Integer > result = new Pair( offspring2, looser );
return result;
}

private Pair< Integer, Integer >
convertIndexToPosition( int index )
{
    return new Pair< Integer, Integer >(
            index % problem.getSideLength(),
            ( index / problem.getSideLength() ) % problem.getSideLength() );
}

/**
 * Distributes memetic parameters values along the terrain's axis.
 * This needs to be expanded to support more than 5x5 terrains
 * @param low is the low end of the scale factors
 * @param high is the upper end of the scale factors
 */
public void distribute_parameters( float low, float high )
{
    float deltaSize = ( high - low ) / ( problem.getSideLength() - 1 );
    float startingValue = low;
    ArrayList<ArrayList<ParametersMTBMA>> param = new ArrayList<ArrayList<ParametersMTBMA>>();
    param.ensureCapacity(problem.getSideLength());
    for (int i = 0; i < problem.getSideLength(); ++i) {
        ArrayList<ParametersMTBMA> tmp = new ArrayList<ParametersMTBMA>();
        tmp.ensureCapacity(problem.getSideLength());
        float y = startingValue + i * deltaSize;
        for (int k = 0; k < problem.getSideLength(); ++k) {
            tmp.add(k, new ParametersMTBMA());
            tmp.get(k).scale_factor = startingValue + k * deltaSize;
            tmp.get(k).scale_factor2 = y;
        }
        param.add(i, tmp);
    }
    problem.setParametersMTBMA(param);
}

/**
 * Returns an array indicating the number of individuals at each location
 * in the population
 */
public int[]
examineCity()
if( problem.getMemeticMode() )
{
    int[] returnValue = new int[solutions.length];
    for( int i = 0; i < solutions.length; ++i )
    {
        returnValue[i] = 0;
    }
    for( int i = 0; i < solutions.length; ++i )
    {
        Pair< Integer, Integer > pos = solutions[i].getPosition();
        int index = pos.getFirst() * problem.getSideLength() + pos.getSecond();
        returnValue[index] += 1;
    }
    return returnValue;
}
else
{
    int[] result = new int[solutions.length];
    Arrays.fill(result, 0);
    return result;
}
/**
* Evolves the next generation of solutions.
*<p>
* The solutions reside on a grid whose edges wrap around forming a torous.
* Every <code>Individual</code> solution is mated with the highest fitness
* of the original parent and both offspring replaces the parent in the
* next generation.
*<p>
* Crossover and mutation values are automatically configured based on the
* location of the <code>Individual</code> on the grid, unless the software
* is set to operate as a cellular genetic algorithm with fixed values.
*/
public void evolve()
{
    System.out.println( "In evolve" );
    Individual kid1, kid2, mate;
    int location, sideLen, xover;
    double mutrt;
    kid1 = new Individual();
    kid2 = new Individual();
    kid1.setProblem(problem);
    kid2.setProblem(problem);
    sideLen = problem.getSideLength();
    if( problem.getMemeticMode() )
    {
        ArrayList<Pair<Integer, Individual>> aux = new ArrayList<Pair<Integer, Individual>>();
        ArrayList<Integer> neighbors = new ArrayList<Integer>();
        //System.out.println("For each city " + System.currentTimeMillis());

// For each city i in population
for (int i = 0; i < solutions.length; ++i) {
    int x = solutions[i].getPosition().getFirst();
    int y = solutions[i].getPosition().getSecond();

    ArrayList<Integer> citizens = new ArrayList<Integer>();

    // Find best neighbour
    if (city_ok.get(x).get(y) == 0) {
        neighbors.add(selection(i).getIndex());
        city_ok.get(x).set(y, i);
    } else {
        int bestN = solutions.length - 1;
        if (neighbors.size() > city_ok.get(x).get(y)) {
            bestN = neighbors.get(city_ok.get(x).get(y));
        }
        neighbors.add(bestN);
    }

    int tournament_size = 3;
    // Apply GENITOR to evolve city
    if (citizens.size() >= 2) {
        if (citizens.size() == 2) {
            tournament_size = 2;
        }
        GENITOR(citizens, tournament_size);
    }
}

ArrayList<ArrayList<Integer>> ok = new ArrayList<ArrayList<Integer>>(5);
city_ok.clear();
city_ok.ensureCapacity(solutions.length);

for (int bigger = 0; bigger < solutions.length; ++bigger) {
    city_ok.add(bigger, new ArrayList<Integer>());
    city_ok.get(bigger).ensureCapacity(solutions.length);
    for (int innerBigger = 0; innerBigger < solutions.length; ++innerBigger) {
        city_ok.get(bigger).add(innerBigger, new Integer(0));
    }
}

//System.out.println("For each mayor " + System.currentTimeMillis());
ArrayList<Integer> Mayors = new ArrayList<Integer>();
for (int i = 0; i < solutions.length; ++i) {
    int x = solutions[i].getPosition().getFirst();
    int y = solutions[i].getPosition().getSecond();

    //System.out.println("Finding Mayor for city at (" + x + "," + y + ")");
    if (city_ok.get(x).get(y) == 0) {
        Integer Mayor = retrieve_best_citizen(i);
        Mayors.add(Mayor);
    }
    int tmp = city_ok.get(x).get(y);
    city_ok.get(x).set(y, tmp + 1);
}

city_ok.clear();
city_ok.ensureCapacity(solutions.length);

for (int bigger = 0; bigger < solutions.length; ++bigger) {
    city_ok.add(bigger, new ArrayList<Integer>());
    city_ok.get(bigger).ensureCapacity(solutions.length);
    for (int innerBigger = 0; innerBigger < solutions.length; ++innerBigger) {
city_ok.get(bigger).add(innerBigger, new Integer(0));
}

for (int i = 0; i < Mayors.size(); ++i) {
    Individual Mayor = solutions[Mayors.get(i)];
    int index = Mayor.getIndex();
    int x = solutions[i].getPosition().getFirst();
    int y = solutions[i].getPosition().getSecond();
    // Find best Neighbouring Mayor
    int bestF = retrieve_best_foreign(Mayors, i);
    Individual parent = solutions[Mayors.get(i)];
    int parentX = solutions[i].getPosition().getFirst();
    int parentY = solutions[i].getPosition().getSecond();
    Individual offspring = new Individual().create_individual(parent.getGenotype(),
        parentX, parentY, parent.getRank());
    offspring.setCode(parent.getCode());
    offspring.setIndex(parent.getIndex());
    if (bestF != Mayors.size() - 1) {
        Individual bestFIndividual = solutions[bestF];
        x = bestFIndividual.getPosition().getFirst();
        y = bestFIndividual.getPosition().getFirst();
        // Perform crossover
        offspring = crossover(Mayor, bestFIndividual, problem.getVQ());
    } else {
        int j = 0;
        int endOfWhile = i;
        do {
            j = (int) myRandom.random_range(myRandom.getSeed(), 0, Mayors.size());
            bestF = j;
        } while (bestF == endOfWhile);
        Individual first = solutions[Mayors.get(i)];
        Individual second = solutions[bestF];
        offspring = crossover(first, second, problem.getVQ());
    }
    // Perform mutation
    offspring = mutate(offspring, problem.getVQ(),
        (float)problem.getMutationRate(i%problem.getSideLength()));
    // System.out.println("Creating phenotype");
    // From this point, individual is born and, thus, has a phenotype
    create_phenotype(offspring, problem.getVQ());
    // System.out.println("Evaluating...");
    problem.evaluate(offspring);
    // System.out.println("Fitness after " + offspring.fitness);
    // Perform local search
    Individual one = solutions[Mayors.get(i)];
    int l = one.getPosition().getFirst();
    int c = one.getPosition().getSecond();
    // Extraindo a taxa de mutaçao do ambiente
ArrayList<Float> s = new ArrayList<Float>();
s.ensureCapacity(2);
s.add(0, problem.getParametersMTBMA().get(l).get(c).scale_factor);
s.add(1, problem.getParametersMTBMA().get(l).get(c).scale_factor2);

//System.out.println("Now perform local search " + System.currentTimeMillis());
for (int z = 0; z < 2; ++z) {
    //as a reminder, phenotype is the mapping between the myVQ.X and I.genotype
    //this creates the map -> newIndividual = myVQ.calc_partition( I, myVQ.X, I.genotype
    ArrayList<ArrayList<Float>> updatedCodebook =
    problem.getVQ().learning_by_lee(s.get(z), 2.0f, offspring.getPhenotype(),
    offspring.getGenotype(), problem.getVQ().X, myRandom);
    offspring.setGenotype( updatedCodebook );
    // Individual's phenotype is modified by learning
    //System.out.println( "Update offspring phenotype" );
    //System.out.println( "update complete" );
}

//System.out.println("Evaluate fitness after local search " +
System.currentTimeMillis());
// Evaluate individual's fitness
offspring.setFitness( problem.evaluate(offspring) );
//offspring.fitness = evaluate(offspring, myVQ);

// Insert offspring into auxiliary population
Pair<Integer, Individual> swap = new Pair<Integer, Individual>(Mayors.get(i),
offspring);
aux.add(swap);

// Reset rank - commented out in Carlos' code
//P.get( i ).rank = 0;
//P[i].rank = 0;

//System.out.println("Update population " + System.currentTimeMillis());
// Update population
for (int i = 0; i < aux.size(); ++i) {
    Individual tmp = solutions[aux.get(i).getFirst()];
    if (tmp.getFitness() < aux.get(i).getSecond().getFitness()) {
        //Yes, we will replace P with the new offspring
        replace( aux.get(i).getFirst(), aux.get(i).getSecond() );
    } else {
        newSolutions[i] = tmp;
        newSolutions[i].setIndex(tmp.getIndex());
    }
}

//update_terrain(best, P, T);
if( problem.getMemeticMode() && problem.getMovementEnabled() ) {
    update_agents_positions(neighbors);
} else {
    for ( int row = 0; row < sideLen; row++ ) {
        

for ( int col = 0; col < sideLen; col++ )
{
    if ( problem.isTerrainMode() )
    {
        // Use Auto-Configured Values
        xover = problem.getCrossoverPoints(col);
        mutrt = problem.getMutationRate(row);
    }
    else
    {
        // Use Default Values
        xover = problem.getCrossoverPoints();
        mutrt = problem.getMutationRate();
    }

    location = row * sideLen + col;
    mate = selection(location);

    if ( xover > 0 )
    {
        crossover(solutions[location], mate, kid1, kid2, xover);
    }
    else
    {
        kid1.setCode(solutions[location].getCode());
        kid2.setCode(mate.getCode());
    }

    if ( mutrt > 0.0 )
    {
        kid1.mutate(mutrt);
        kid2.mutate(mutrt);
    }

    problem.evaluate(kid1);
    problem.evaluate(kid2);

    replace(location, kid1, kid2);
}

// Replace Current Generation with Next Generation
for ( int i = 0; i < solutions.length; ++i )
{
    solutions[i] = newSolutions[i];
}

/*
for ( int i = 0; i < solutions.length; ++i )
{
    System.out.println(solutions[i].getPosition().getFirst() + "--" + solutions[i].getPosition().getSecond());
}
*/

/**
 * Returns a copy of the <code>Individual</code> solution in the current <code>Population</code> with a fitness closest to the <code>Problem</code> target value.
 */

/**
 * Method to evolve the city
* @param citizens The number of citizens to compete
* @param tsize The size of the tournament
*/

private void GENITOR(ArrayList<Integer> citizens, int tsize) {
    int num_offspring = citizens.size() - 1;
    ArrayList<Integer> my_city = new ArrayList<Integer>);
    Individual[] newP = new Individual[problem.getPopulationSize()];

    ArrayList<Pair<Integer, Individual>> new_city = new ArrayList<Pair<Integer,
    Individual>>()
    
    // Initialize new City
    System.out.println("Citizens size "+ citizens.size() + "+")
    //Individual mayor = P.get( citizens.get( 0 ) );
    int mayor = citizens.get( 0 );
    for (int i = 0; i < citizens.size(); ++i) {
        int tmp = citizens.get( i );
        my_city.add(tmp);
        Individual newIndividual = solutions[my_city.size()-1];
        if (newIndividual.getFitness() > solutions[mayor].getFitness()) {
            mayor = solutions.length-1;
        }
    }

    // Generate |P| - Ni offspring
    for (int j = 0; j < num_offspring; ++j) {
        Individual parent = solutions[parents.get(0)];
        Individual offspring = new Individual();
        offspring.create_individual(parent.getGenotype(), parent.getPosition().getFirst(),
        parent.getPosition().getSecond(), parent.getRank());
        offspring.setProblem(parent.getProblem());
        offspring.setCode(parent.getCode());
        offspring.setIndex(parent.getIndex());
        int looser = 0;
        Pair< Individual, Integer > resultOffspringGenitor =
            crossover(solutions[parents.get(0)], solutions[parents.get(1)], looser, problem.getVQ());
        Individual offspringGenitor = resultOffspringGenitor.getFirst();
        looser = resultOffspringGenitor.getSecond();
        // From this point, individual is born and, thus, has a phenotype
        create_phenotype(offspringGenitor, problem.getVQ());
        offspring.setFitness(problem.evaluate(offspringGenitor));
        // Perform local search
        int l = parent.getPosition().getFirst();
        int c = parent.getPosition().getSecond();

        // Extraindo a taxa de muta do ambiente
        ArrayList<Float> s = new ArrayList<Float>();
        //s.setSize(2);
        s.ensureCapacity(2);
        s.add(problem.getParametersMTBMA().get(l).get(c).scale_factor);
        s.add(problem.getParametersMTBMA().get(l).get(c).scale_factor2);

        offspringGenitor = mutate(offspringGenitor, problem.getVQ(),
            (float)problem.getMutationRate(parents.get(0)))
for (int m = 0; m < 2; ++m) {
    problem.getVQ().learning_by_lee(s.get(m), 2, offspring.getPhenotype(),
    offspring.getGenotype(), problem.getVQ().X, myRandom);
    // Individual's phenotype is modified by learning
    create_phenotype(offspringGenitor, problem.getVQ());
}

// Evaluate individual's fitness
offspringGenitor.setFitness(problem.evaluate(offspringGenitor));
Pair<Integer, Individual> swap = new Pair<Integer, Individual>(looser, offspring);
new_city.add(swap);

// Replace if slightly different than looser
    if (fabs(offspring.psnr - looser->psnr) > .05)
    }

for (int i = 0; i < new_city.size(); ++i) {
    double u = new Double(0.0);
    u = myRandom.value_range(myRandom.getSeed(), 0.0f, 1.0f);
    //if (new_city[i].first->fitness < new_city[i].second.fitness)
    {//
        Pair<Integer, Individual> tmp = new_city.get(i);
        Integer tmpIter = tmp.getFirst();
        Individual mayorIndividual = solutions[mayor];
        Individual tmpInd = solutions[tmpIter];
        int l = tmpInd.getPosition().getFirst();
        int c = tmpInd.getPosition().getSecond();
        //cout << "Replacing\n";
        //system("pause");
        if (tmpIter != mayor || tmpInd.getFitness() > mayorIndividual.getFitness()) {
            Individual newIndividual = new_city.get(i).getSecond();
            int tmpIter2 = new_city.get(i).getFirst();
            Individual originalIndividual = solutions[tmpIter2];
            originalIndividual = newIndividual;
            solutions[tmpIter2] = newIndividual;
        }
    }
    }

/**
 * Method finds all citizen positions for a given population and terrain
 * @param i Starting position
 * @return The positions of each citizen
 */
ArrayList<Integer> get_citizens(int i) {
    int d = problem.getSideLength();
    int pos;
    Pair<Integer, Integer> coordenada = solutions[i].getPosition();
    ArrayList<Integer> vizinhos = new ArrayList<Integer>();
    pos = -1;
    do {
        pos = get_neighbor(coordenada, pos + 1, i);
        if (pos != -1) {
            vizinhos.add(pos);
        }
    } while (pos != -1);
    return vizinhos;
}
*/
* Method finds the foreign mayors
* @param P The population to search
* @param T The terrain to search
* @param i The position we started from
* @return The positions of the foreign mayors
*/
ArrayList<Integer> get_foreigns(int i) {
    int d = problem.getSideLength();
    int pos = new Integer(0);
    int x = new Integer(0);

    Pair<Integer, Integer> coordenada = solutions[i].getPosition();
    Pair<Integer, Integer> csul = new Pair<Integer, Integer>(
        (coordenada.getFirst() + 1) % d, coordenada.getSecond());
    Pair<Integer, Integer> cleste = new Pair<Integer, Integer>(
        coordenada.getFirst(),
        (coordenada.getSecond() + 1) % d);
    x = coordenada.getFirst() > 0 ? coordenada.getFirst() - 1 : d - 1;
    Pair<Integer, Integer> cnorte = new Pair<Integer, Integer>(x, coordenada.getSecond());
    x = coordenada.getSecond() > 0 ? coordenada.getSecond() - 1 : d - 1;
    Pair<Integer, Integer> coeste = new Pair<Integer, Integer>(
        coordenada.getFirst(), x);

    ArrayList<Pair<Integer, Integer>> vizinhanca = new ArrayList<Pair<Integer, Integer>>(4);
    vizinhanca.add(cnorte);
    vizinhanca.add(cleste);
    vizinhanca.add(csul);
    vizinhanca.add(coeste);

    ArrayList<Integer> vizinhos = new ArrayList<Integer>();
    for (int j = 0; j < 4; ++j) {
        pos = -1;
        do {
            pos = get_neighbor(vizinhanca.get(j), pos + 1, i);
            if (pos != -1) {
                vizinhos.add(pos);
            }
        } while (pos != -1);
    }
    return vizinhos;
}

/**
* Method returns the neighbor from a population/terrain given the starting coordinate
* @param coord The position to look for the neighbor
* @return The position of the neighbor or -1 if not found
*/
int get_neighbor(Pair<Integer, Integer> coord, int pos, int i) {
    int popsize = solutions.length;
    for (int j = pos; j < popsize; ++j) {
        if (j != i && solutions[j].getPosition().getFirst() == coord.getFirst() &&
            solutions[j].getPosition().getSecond() == coord.getSecond()) {
            return j;
        }
    }
    return -1;
}

/**
* Method to seed the individual's codebook
*/
* @param id The problem to solve
* @param myMTBMA The MTMBA to receive the population
* @param myVQ The VQ that will be used to solve the population
* @return
*/
private Individual loadCodebook(Individual orig, String id, int index, VQ myVQ)
{
    index += 1;
    String path = id + "" + id + ".dic." + myVQ.N + "." + index + ".dat";
    //System.out.println("Path for pop is " + path);
    ArrayList<ArrayList<Float>> W = new ArrayList<ArrayList<Float>>();
    ArrayList<Float> w = new ArrayList<Float>();
    int j = 1;
    try {
        File file = new File(path);
        Scanner input = new Scanner(file);
        while (input.hasNext()) {
            String nextToken = input.next();
            float x = Float.parseFloat(nextToken);
            w.add(x);
            if (j % myVQ.K == 0) {
                ArrayList<Float> tmp = new ArrayList<Float>();
                //Collections.copy(tmp, w);
                tmp.ensureCapacity(w.size());
                tmp.addAll(w);
                W.add(tmp);
                w.clear();
            }
            ++j;
        }
        input.close();
    } catch (FileNotFoundException e) {
        System.err.println("Error when opening " + path + ".");
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println("Exception while reading the file" + ioe);
    }
    orig.setGenotype(W);
    return orig;
}

/**
 * Method to load codebooks into the VQ
 * @param id Identifier of the codebook
 * @param myVQ The VQ to receive the codebook
 * @return
 */
public ArrayList<ArrayList<Float>> load_signal(String id, VQ myVQ) {
    String path = id + "/" + id + ".dat";
    ArrayList<ArrayList<Float>> W = new ArrayList<ArrayList<Float>>();
    ArrayList<Float> w = new ArrayList<Float>();
    int j = 1;
    try {
        FileInputStream fstream = new FileInputStream(path);
        DataInputStream in = new DataInputStream(fstream);
        while (in.available() != 0) {
            String nextToken = in.readUTF();
            w.add(Float.parseFloat(nextToken));
            if (j % myVQ.K == 0) {
                ArrayList<Float> tmp = new ArrayList<Float>();
                //Collections.copy(tmp, w);
                tmp.ensureCapacity(w.size());
                tmp.addAll(w);
                W.add(tmp);
                w.clear();
            }
            ++j;
        }
        in.close();
    } catch (FileNotFoundException e) {
        System.err.println("Error when opening " + path + ".");
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println("Exception while reading the file" + ioe);
    }
    return W;
}
String input = in.readLine();
float x = Float.parseFloat(input);

w.add(x);
if (j % myVQ.K == 0) {
    ArrayList<Float> tmp = new ArrayList<Float>();
    //tmp.setSize(w.size());
    //Collections.copy(tmp, w);
    tmp.ensureCapacity(w.size());
    tmp.addAll(w);
    X.add(tmp);
    //System.out.println( "X -- " + X.firstElement().firstElement() + "---" + X.get(0).get(0) + " --- " + x );
    //System.out.println( X );
    w.clear();
}

++j;

in.close();
}
)
catch (FileNotFoundException e) {
    System.err.println("Error when opening " + path + ".");
    System.exit(1);
}
catch (IOException ioe) {
    System.err.println("Exception while reading the file" + ioe);
}

return X;
}

/**
 * Method determines if an individual can move to a new position given the maximum
 * citizens per location in the terrain
 * @param I The position of the individual wanting to move
 * @param P The population to examine
 * @param T The terrain to examine
 * @param destiny The desired new position
 * @return vector<> (bool moved?, updated population, updated terrain )
 */
ArrayList< Object > moveTo(Integer I, Pair<Integer,Integer> destiny) /* Move agent on
terrain */
{
    Boolean moved = false;
    ArrayList< Object > returnValues = new ArrayList< Object >();
    int index = I;
    int l = destiny.getFirst();
    int c = destiny.getSecond();
    int a = solutions[I].getPosition().getFirst();
    int b = solutions[I].getPosition().getSecond();

    if(qtd.get(l).get(c) < problem.getMaxPerCell() )
    {
        qtd.get( a ).set( b, qtd.get( a ).get( b ) - 1);
        solutions[index].setPosition( destiny );
        positions.set( index, destiny );
        qtd.get( l ).set( c, qtd.get( l ).get( c ) + 1 );
        moved = true;
        returnValues.add( moved );
        return returnValues;
    }
    returnValues.add( moved );
    return returnValues;
}
/** 
 * Method to mutate a given individual 
 * @param I is the individual to mutate 
 * @return the resulting mutated individual 
 */
Individual mutate(Individual I, VQ myVQ, float mut_rate) {
    Individual resultingIndividual = I;
    for (int i = 0; i < myVQ.N; ++i) {
        float u = (float)problem.nextDouble();
        //   float u = myRandom.value_range(myRandom.getSeed(), 0, 1);
        if (u < mut_rate) {
            for (int j = 0; j < myVQ.K; ++j) {
                float m = (float)myRandom.gauss_box_muller(0.0, 1.0);
                double x = (int) (resultingIndividual.getGenotype().get(i).get(j) + m);
                x = (x < 0) ? 0 : ((x > myVQ.dsup) ? myVQ.dsup : x);
                resultingIndividual.getGenotype().get(i).set(j, (float) x);
            }
        }
    }
    return resultingIndividual;
}
/** 
 * Determines the best individual within a given (sub)population. 
 * @param P A given (sub)population; 
 * @param T The terrain on which the given P is distributed; 
 * @param index The index (ID) of the given (sub)population (i.e., city); 
 * @return The position of the best individual. 
 */
Integer retrieve_best_citizen(int index) {
    // "Vizinhos" means "neighbors" in portuguese :) 
    // This will retrieve the indexes (IDs) to every 
    // citizen within a city, except 
    ArrayList<Integer> vizinhos = get_citizens(index);
    vizinhos.add(index);
    int vsize = vizinhos.size();
    if (vsize == 1) {
        return index;
    }
    int best = vizinhos.get(0);
    int tmp = best;
    Individual bestIndividual = solutions[best];
    for (int j = 1; j < vsize; ++j) {
        if (solutions[vizinhos.get(j)].getFitness() > bestIndividual.getFitness()) {
            best = vizinhos.get(j);
        }
    }
    return best;
}
/** 
 * Find best foreign Mayor 
 * @param P A given (sub)population; 
 * @param T The terrain on which the given P is distributed; 
 * @param index The index (ID) of a given individual; 
 */
* @return the position of the best foreign mayor

```
Integer retrieve_best_foreign(ArrayList<Integer> C, int index) {
    ArrayList<Individual> P = new ArrayList<Individual>();
    ArrayList<Integer> localC = C;
    for (int i = 0; i < C.size(); ++i) {
        Individual newIndividual = solutions[localC.get(i)];
        P.add(newIndividual);
    }
    //cout << "Mayors found: " << C.size() << endl;
    ArrayList<Integer> vizinhos = get_foreigns(index);
    int vsize = vizinhos.size();
    //cout << "Foreigns Mayors found: " << vsize << endl;
    if (vsize == 0) {
        return solutions.length - 1;
    }
    // population vizinhos_pop;
    int best = localC.get(0);
    Individual bestIndividual = new Individual();
    Individual localCIndividual = new Individual();
    bestIndividual = solutions[vizinhos.get(0)];
    //cout << vizinhos[0] << " ";
    if( vsize > localC.size() ) {
        vsize = localC.size();
    }
    for (int j = 1; j < vsize; ++j) {
        localCIndividual = solutions[localC.get(j)];
        if (localCIndividual.getFitness() > bestIndividual.getFitness()) {
            best = localC.get(j);
        }
    }
    return best;
}
```

```
private int
getNorth( int index )
{
    int popSize = problem.getPopulationSize();
    int sideLen = problem.getSideLength();
    int top = 0;
    // Determine top location
    if { index < sideLen }
    top = popSize - sideLen + index;
    else
    top = index - sideLen;
    return top;
}
```

```
private int
getSouth( int index )
{
    int popSize = problem.getPopulationSize();
    int sideLen = problem.getSideLength();
    int bottom = 0;
    //Determine bottom location
    if ( index >= (popSize - sideLen) )
    bottom = index + sideLen - popSize;
    else
bottom = index + sideLen;
return bottom;
}

private int getWest( int index ) {
    int popSize = problem.getPopulationSize();
    int sideLen = problem.getSideLength();
    int left = 0;
    // Determine Left position
    if ( (index % sideLen) == 0 )
        left = index - 1 + sideLen;
    else
        left = index - 1;
    return left;
}

private int getEast( int index ) {
    int popSize = problem.getPopulationSize();
    int sideLen = problem.getSideLength();
    int right = 0;
    // Determine Right Location
    if ( ((index + 1) % sideLen) == 0 )
        right = index + 1 - sideLen;
    else
        right = index + 1;
    return right;
}

/**
 * Method to move individuals on the terrain toward the mayor
 * @param vizinhos position of the neighbors
 * @param P The population to examine
 * @param T The terrain to examine
 */
private void update_agents_positions( ArrayList<Integer> vizinhos ) {
    System.out.println( "Starting update agents positions" );
    //typedef pair<pair<int,int>,pair<int,int>> move_coord;
    //typedef pair<Individual_Ptr,Individual_Ptr> move;
    //typedef pair<move_coord, move> movement;
    ArrayList< Pair< Pair< Pair< Integer, Integer >, Pair< Integer, Integer > >, Pair< Integer, Integer > > > M =
    new ArrayList< Pair< Pair< Pair< Integer, Integer >, Pair< Integer, Integer > >, Pair< Integer, Integer > > >();
    int p = 0;
    for( int i = 0; i < solutions.length; ++i, ++p)
    {
        if(( vizinhos.get(i) != solutions.length - 1) && solutions[i].getPosition() != solutions[vizinhos.get(i)].getPosition())
        {
            if( solutions[i].getFitness() < solutions[vizinhos.get(i)].getFitness())
            {
                int index = vizinhos.get(i);
Pair< Pair< Integer, Integer >, Pair< Integer, Integer > > pos =
    new Pair< Pair< Integer, Integer >, Pair< Integer, Integer > > (solutions[i].getPosition(), solutions[index].getPosition() );
Pair< Integer, Integer > location = new Pair< Integer, Integer > ( p, vizinhos.get( i ) );
Pair< Integer, Integer > newElement =
    new Pair< Pair< Pair< Integer, Integer >, Pair< Integer, Integer > >, Pair< Integer, Integer > > ( pos, location );
M.add( newElement );
}
}
else {
    double potentialToMigrate = .2;
    double u = myRandom.value_range(myRandom.getSeed(), 0.0f, 1.0f);
    if(vizinhos.get(i) == solutions.length-1) {
        potentialToMigrate = 1.0; // It will migrate!
    }
    // If best neighbour is within the same city, it has a chance of migrating...
    if(solutions[vizinhos.get(i)].getFitness() != solutions[i].getFitness()) {
        if (u < potentialToMigrate) {
            ArrayList< Object > moveToResponse = new ArrayList< Object >();
            Boolean moved = false;
            int max = 16;
            int localK = 0;
            do {
                u = myRandom.value_range(myRandom.getSeed(), 0.0f, 1.0f);
                int I = i;
                if (u < 0.25) {
                    Pair< Integer, Integer > north = solutions[getNorth(i)].getPosition();
                    moveToResponse = moveTo(I,north);
                } else if (u >= 0.25 && u < 0.5) {
                    Pair< Integer, Integer > west = solutions[getWest(i)].getPosition();
                    moveToResponse = moveTo(I,west);
                } else if (u >= 0.5 && u < 0.75) {
                    Pair< Integer, Integer > east = solutions[getEast(i)].getPosition();
                    moveToResponse = moveTo(I,east);
                } else {
                    Pair<Integer,Integer> south = solutions[getSouth(i)].getPosition();
                    moveToResponse = moveTo(I,south);
                }
                ++localK;
                moved = (Boolean)moveToResponse.get(0);
            } while (!moved && localK < max);
        } else if (u < potentialToMigrate) {
            // Moving Mayor
            ArrayList< Object > moveToResponse = new ArrayList< Object >();
            Boolean moved = false;
            int max = 16;
            int k = 0;
do
{
    u = myRandom.value_range(myRandom.getSeed(), 0.0f, 1.0f);
    int I = i;
    if (u < 0.25)
    {
        Pair< Integer, Integer > north = solutions[getNorth(i)].getPosition();
        moveToResponse = moveTo(I,north);
    }
    else if (u >= 0.25 && u < 0.5)
    {
        Pair< Integer, Integer > west = solutions[getWest(i)].getPosition();
        moveToResponse = moveTo(I,west);
    }
    else if (u >= 0.5 && u < 0.75)
    {
        Pair< Integer, Integer > east = solutions[getEast(i)].getPosition();
        moveToResponse = moveTo(I,east);
    }
    else
    {
        Pair<Integer,Integer> south = solutions[getSouth(i)].getPosition();
        moveToResponse = moveTo(I,south);
    }
    ++k;
    moved = (Boolean)moveToResponse.get(0);
} while (!moved && k < max);
}

int msize = M.size();
for(int i = 0; i < msize; ++i)
{
    Boolean moved = false;
    int a = M.get( i ).getFirst().getFirst().getFirst();
    int b = M.get( i ).getFirst().getFirst().getSecond();
    int l = M.get( i ).getSecond().getFirst().getFirst();
    int c = M.get( i ).getSecond().getSecond().getSecond();
    Pair< Integer, Integer > destiny = new Pair< Integer, Integer >( l, c );
    int I = M.get( i ).getSecond().getFirst();
    double potentialToMove = 0.2;
    double u = myRandom.value_range( myRandom.getSeed(), 0.0f, 1.0f );
    ArrayList< Object > resultOfMove = new ArrayList< Object >();
    resultOfMove = moveTo( I, destiny );
    moved = (Boolean)resultOfMove.get( 0 );
    //cout << "(" << a << "," << b << ") ";
    if( moved == true )
    {
        System.out.println( "Moved to destiny (" + l + "," + c + ")" );
    }
    else if (u < potentialToMove)
    {
        Pair< Integer, Integer > C = new Pair< Integer, Integer >(l,c);
        int index = l * problem.getSideLength() + c;
        Pair< Integer, Integer > north = convertIndexToPosition( getNorth(index) );
        Pair< Integer, Integer > south = convertIndexToPosition( getSouth(index) );
        Pair< Integer, Integer > west = convertIndexToPosition( getWest(index) );
        Pair< Integer, Integer > east = convertIndexToPosition( getEast(index) );
        Individual tmp = solutions[i];
// We are at south
if (tmp.getPosition() == south)
{
    resultOfMove = moveTo(I, north);
    if( (Boolean)resultOfMove.get( 0 ) == false )
    {
        System.out.println( "Moved to north" );
    } else {
        resultOfMove = moveTo(I, east);
        if( (Boolean)resultOfMove.get( 0 ) == true )
        {
            System.out.println( "Moved to east" );
        } else {
            resultOfMove = moveTo(I, west);
            if( (Boolean)resultOfMove.get( 0 ) == true )
            {
                System.out.println( "Moved to west" );
            } else {
                System.out.println( "Not moved" );
            }
        }
    }
} // We are at west
else if (tmp.getPosition() == west)
{
    resultOfMove = moveTo(I, east);
    if( (Boolean)resultOfMove.get( 0 ) == false )
    {
        System.out.println( "Moved to east" );
    } else {
        resultOfMove = moveTo(I, south);
        if( (Boolean)resultOfMove.get( 0 ) == true )
        {
            System.out.println( "Moved to south" );
        } else {
            resultOfMove = moveTo(I, north);
            if( (Boolean)resultOfMove.get( 0 ) == true )
            {
                System.out.println( "Moved to north" );
            } else {
                System.out.println( "Not moved" );
            }
        }
    }
} // We are at north
else if (tmp.getPosition() == north)
{
    resultOfMove = moveTo(I, south);
    if( (Boolean)resultOfMove.get( 0 ) == false )
    {
        System.out.println( "Moved to south" );
    } else {
        resultOfMove = moveTo(I, east);
        if( (Boolean)resultOfMove.get( 0 ) == true )
        {
            System.out.println( "Moved to east" );
        } else {
            resultOfMove = moveTo(I, west);
            if( (Boolean)resultOfMove.get( 0 ) == true )
            {
                System.out.println( "Moved to west" );
            } else {
                System.out.println( "Not moved" );
            }
        }
    }
}
else
{
    resultOfMove = moveTo(I, west);
    if( (Boolean)resultOfMove.get(0) == true )
    {
        System.out.println("Moved to west");
    }
    else
    {
        resultOfMove = moveTo(I, east);
        if( (Boolean)resultOfMove.get(0) == true )
        {
            System.out.println("Moved to east");
        }
        else
        {
            System.out.println("Not moved");
        }
    }
}
// We are at east
else if (tmp.getPosition() == east)
{
    resultOfMove = moveTo(I, west);
    if( (Boolean)resultOfMove.get(0) == false )
    {
        System.out.println("Moved to west");
    }
    else
    {
        resultOfMove = moveTo(I, north);
        if( (Boolean)resultOfMove.get(0) == true )
        {
            System.out.println("Moved to north");
        }
        else
        {
            resultOfMove = moveTo(I, south);
            if( (Boolean)resultOfMove.get(0) == true )
            {
                System.out.println("Moved to south");
            }
            else
            {
                System.out.println("Not moved");
            }
        }
    }
}
else
{
    System.out.println("ERROR (" + destiny.getFirst() + "," + destiny.getSecond() + ")");
}
A.10 – Changes to Problem.java

```java
public abstract class Problem implements Cloneable {
    private boolean siftMode;
    private boolean terrainMode;
    private double mutationRate;
    private double mutationTable[];
    private double target;
    private int chromosomeLength;
    private int crossoverPoints;
    private int crossoverTable[];
    private int experiments;
    private int generations;
    private int populationSize;
    private int sideLength;
    private int maxPerCell;
    private long seed;
    private Random randomGen;
    private boolean memeticMode;
    private boolean movementEnabled;
    private float lowerScaleRange = 1.1f;
    private float upperScaleRange = 1.4f;

    /**
     * The constructor should be used to set default parameter values.
     */
    public Problem() {
        seed = System.currentTimeMillis();
        randomGen = new Random(seed);
        memeticMode = false;
        movementEnabled = false;
        lowerScaleRange = new Float( 1.0 );
        upperScaleRange = new Float( 1.4 );
        maxPerCell = 1;
    }

    /**
     * Returns a new <code>Problem</code> that is a deep copy of this
     * <code>Problem</code>.  The clone's <code>Random</code> number
     * generator will be reset to the initial state.
     */
    public Object clone() {
        Problem c = null;
        try {
            c = (Problem) super.clone();
            c.setChromosomeLength(chromosomeLength);
            c.setCrossoverPoints(crossoverPoints);
            c.setExperiments(experiments);
            c.setGenerations(generations);
            c.setMutationRate(mutationRate);
            c.setTarget(target);
        } catch (CloneNotSupportedException e) {
            // handle exception
        }
        return c;
    }
}
```
c.setSeed(seed);
c.setSideLength(sideLength);
c.setSiftMode(siftMode);
c.setTerrainMode(terrainMode);
c.setUpperScaleFactor( upperScaleRange );
c.setLowerScaleFactor( lowerScaleRange );
} catch (CloneNotSupportedException e) {
  System.err.println(e.getMessage());
}

return c;
}
/**
 * For algorithms with local search, return the tuning parameter #1
 */
public abstract String
getFactor1();
/**
 * For algorithms with local search, return the tuning parameter #2
 */
public abstract String
getFactor2();
/**
 * Abstract method to enable specifying the image to use in the
 * MTBMA specialization
 */
public abstract String
getImageToUtilize();
/**
 * Returns the lower end of the scale factor
 */
public Float
getLowerScaleFactor()
{
  return this.lowerScaleRange;
}
/**
 * Returns the maximum individuals per cell on the terrain for tbma algos.
 */
public Integer
getMaxPerCell()
{
  return this.maxPerCell;
}
/**
 * Returns whether or not the problem implements terrain based memetic algorithms
 */
public Boolean
getMemeticMode()
{
  return memeticMode;
}
/**
 * Returns if the problem permits movement of individuals
 */
public Boolean
getMovementEnabled()
if( !memeticMode )
{
    return false;
}
else
{
    return movementEnabled;
}

/**
 * Returns the current distributed parameter list of scale factor
 * values utilized in MTBMA code.
 */
public abstract ArrayList< ArrayList< ParametersMTBMA > >
getParametersMTBMA();

/*
 * Return the scale factor 1 for a given index
 */
public float
getScaleFactor1Points(int index)
{
    if ( terrainMode )
    {
        int x = index % sideLength;
        int y = ( index - x ) / sideLength;
        ArrayList< ArrayList< ParametersMTBMA > > params = new ArrayList< ArrayList<
ParametersMTBMA > >();
        getParametersMTBMA(); //System.out.println("Index " + index + " Returning the value for " + x + "," + y + "+ scale);
        float scale = params.get(x).get(y).scale_factor;
        return scale;
    } else
    {
        return 0.0f;
    }
}

/*
 * Return the scale factor 1 for a given index
 */
public float
getScaleFactor2Points(int index)
{
    if ( terrainMode )
    {
        int x = index % sideLength;
        int y = ( index - x ) / sideLength;
        ArrayList< ArrayList< ParametersMTBMA > > params = new ArrayList< ArrayList<
ParametersMTBMA > >();
        getParametersMTBMA();
        float scale = params.get(x).get(y).scale_factor2;
        return scale;
    } else
    {
        return 0.0f;
    }
}

/**
 * Returns the upper end of the scale factor
 */
public Float getUpperScaleFactor() {
    return this.upperScaleRange;
}

/**
 * Get a problem's vector quantizer
 */
public abstract VQ getVQ();

/**
 * Returns true if the genetic algorithm sifts its terrain tables to
 * smoothly distribute parameter values within the toroidal tables.
 */

/**
 * Abstract method to enable specifying the image to use in the
 * MTBMA specialization
 */
public abstract void setImageToUtilize(String image);

/**
 * Sets the lower end of the scale factor
 */
public void setLowerScaleFactor(Float lower) {
    this.lowerScaleRange = lower;
}

/**
 * Sets the maximum number of individuals that may exist at any given
 * point in the terrain for tbma.
 */
public void setMaxPerCell(int value) {
    if(!memeticMode) {
        this.maxPerCell = 1;
        //System.out.println("Max is 1");
    } else {
        this.maxPerCell = value;
        //System.out.println("Max is value " + value);
    }
}

/**
 * Sets whether or not the problem implements terrain based memetic algorithms
 */
public void setMemeticMode(Boolean memeticEnabled) {
    this.memeticMode = memeticEnabled;
}

/**
 * Returns the current distributed parameter list of scale factor
 * values utilized in MTBMA code.
 */
public abstract void
setParametersMTBMA( ArrayList< ArrayList< ParametersMTBMA > > newParameters );

/**
 * Permits setting whether or not the problem will support movement of
 * individuals. Only enabled if the algorithm supports thma.
 */
public void
setMovementEnabled( Boolean value )
{
    System.out.println("Being asked to set movement to " + value);
    if( !memeticMode || value == false )
    {
        this.movementEnabled = false;
    }
    else
    {
        this.movementEnabled = true;
    }
}

/**
 * Sets the lower end of the scale factor
 */
public void
setUpperScaleFactor( Float upper )
{
    this.upperScaleRange = upper;
}
import java.math.BigInteger;
/**
 * @author brad
 */
public class RandomMTBMA {

    private long seed;
    static private int NITER = 4;
    static private long jflone = 0x3f800000;
    static private long jflmsk = 0x007fffff;
    static private long[] cOne = {0xbaa96887L & 0xffffffffL, 0x1e17d32cL & 0xffffffffL,
                                  0x03bcdc3cL & 0xffffffffL, 0x0f33d1b2L & 0xffffffffL,
                                  0x03bcdc3cL & 0xffffffffL, 0x0f33d1b2L & 0xffffffffL};
    static private long[] cTwo = {0x4b0f3b58L & 0xffffffffL, 0xe874f0c3L & 0xffffffffL,
                                   0x6955c5a6L & 0xffffffffL, 0x55a7ca46L & 0xffffffffL};
    static private long idums = 0;

    public void init() {
        seed = 0;
    }

    public void setSeed(long value) {
        seed = value;
    }

    public long getSeed() {
        return seed;
    }

    public long psdes(long lword, long irword) {
        int i = 0;
        long ia = 0;
        long ib = 0;
        long iswap = 0;
        long itmph = 0;
        long itmpl = 0;
        lword = lword & 0xffffffffL;
        irword = irword & 0xffffffffL;
        for (i = 0; i < NITER; i++) {
            ia = (iswap = (irword)) ^ cOne[i];
            itmpl = ia & 0xffff;
            itmph = ia >> 16;
            ib = itmpl * itmph + ~(itmph * itmph);
            irword = (lword)^((ia = (ib >> 16) | (ib & 0xffff) << 16))^cTwo[i] + itmpl * itmph;
            lword = iswap;
        }
        return (irword & 0xffffffffL);
    }

    public float ran4(long idum) {
        long irword = 0;
        long itemp = 0;
        long lword = 0;
        if (idum < 0) {
            idums = (long) -(idum);
            idum = 1;
            setSeed(1);
        }
    }
irword = idum & 0xffffffffL;
lword = idums;
irword = psdes(lword, irword);
itemp = jflone | (jflmsk & irword);
itemp = itemp & 0xffffffffL;
setSeed(++idum);
int abc = (int) itemp;
float returnValue2 = Float.intBitsToFloat(abc) - 1.0f;
//((float)itemp) - 1.0f;
//float returnValue2 = Float.intBitsToFloat( returnValue );
//return (float)itemp-1.0f;
return returnValue2;
}

int random_range(double seed, int min, int max) {
    int num = 0;
    num = (int) (ran4(getSeed()) * (max - min)) + min;
    return num;
}

float value_range(double seed, float min, float max) {
    float num = 0.0f;
    num = (float) ((ran4(getSeed()) * (max - min)) + min);
    return num;
}

double random_range(long seed, double range) {
    return ran4(seed) * (2 * range) - range;
}

double gauss_box_muller(double mean, double deviation) {
    return deviation * (Math.sqrt(-2.0 * Math.log(ran4(getSeed())))
    * Math.cos(2.0 * 3.1415 * ran4(getSeed()))) + mean;
}

double exp_random(double rate) {
    return -Math.log(value_range(getSeed(), 0.0f, 1.0f)) / rate;
}
A.12 – New File VQ.java

/**
 * @author brad
 */

public class VQ {

/**
 * Private variable storing the maximum value for FLOAT
 */
private Double MAX_FLOAT = Double.longBitsToDouble(0x7fefe00000000000L); // The following are the typedefs from Carlos' original code // Left here for reference...
//public Vector<Float> Codevector = new Vector<Float>();
//public Vector<Vector<Float>> Signal = new Vector<Vector<Float>>();
//public ListIterator Codevector_Ptr = Codevector.listIterator();
//public Vector<Object> Voronio_cell = new Vector<Object>();
//public Map<ListIterator, Vector<ListIterator>> Partition;

/**
 * Codebook size
 */
public int N = new Integer(0);

/**
 * Dimension of the vector quantizer
 */
public int K = new Integer(0);

/**
 * Domain lower bound
 */
public double dinf = new Double(0.0);

/**
 * Domain upper bound
 */
public double dsup = new Double(0.0);

/**
 * Vector of signals
 */
public ArrayList<ArrayList<Float>> X = new ArrayList<ArrayList<Float>>();

/**
 * Find the sum of the differences squared
 * @param best is the best citizen
 * @param x
 * @param y
 * @return the sum of the differences squared
 */
public float quadratic_ED(double best, ArrayList<Float> x, ArrayList<Float> y) {
    int size = x.size();
    float sum = 0f;
    for (short i = 0; sum < best && i < size; ++i) {
        float diff = x.get(i) - y.get(i);
        sum += diff * diff;
    }

    return sum;
}
/**
 * Method examines the population to find the neighbour closest to a given position
 * @param x is the codebook
 * @param C is the genotype for the individual
 * @return position and sum of diffs squared for the neighbour
 */
public Pair<Integer, Float> nearest_neighbour(ArrayList<Float> x, ArrayList<ArrayList<Float>> C) {
    int counter = 0;
    int position = 0;
    ListIterator it = C.listIterator();
    ListIterator nn = C.listIterator();
    float qED = quadratic_ED(MAX_FLOAT, x, C.get(0));
    float bqED = qED;
    for (it.next(); it.hasNext();)
    {
        qED = quadratic_ED(bqED, x, (ArrayList<Float>) it.next());
        if (bqED > qED)
        {
            bqED = qED;
            nn = it;
            position = counter;
        }
        ++counter;
    }
    Pair<Integer, Float> returnValue = new Pair(position, bqED);
    return returnValue;
}

/**
 * Method to iterate through an individual's codebook and print the values
 * @param C the vector of values to print
 */
public void print_codevector(ArrayList<Float> C) {
    System.out.println("C<");
    ListIterator it = C.listIterator();
    while (it.hasNext()) {
        System.out.println(it.next() + " ");
    }
    System.out.println(">");
}

/**
 * Method to print out an individual's codebook
 * @param W The codebook of interest
 */
public void print_codebook(ArrayList<ArrayList<Float>> W) {
    ListIterator it = W.listIterator();
    System.out.println("W[");
    while (it.hasNext()) {
        System.out.print("\t");
        print_codevector((ArrayList<Float>) it.next());
    }
    System.out.println("]");
}

/**
 * Method to map codebook and values to the phenotype
 * @param orig the original individual to include the new phenotype
 * @param X The codebook for the Individual
 * @param W The genotype for the individual
 * @return The orig Individual with the new phenotype
 */
public Individual calc_partition(Individual orig, Vector<Vector<Float>> X, Vector<Vector<Float>> W) {
  public Individual calc_partition(Individual orig, ArrayList<ArrayList<Float>> X) {
    Map<Integer, ArrayList<Integer>> P = orig.phenotype;
    ArrayList<ArrayList<Float>> W = orig.getGenotype();
    ListIterator it = X.listIterator();
    Map<Integer, ArrayList<Integer>> newp = new HashMap<Integer, ArrayList<Integer>>();
    int counter = 0;
    float q_err = .0f;
    //System.out.println("Start mapping " + System.currentTimeMillis());
    while (it.hasNext()) {
      //Gives the position in W where the nearest_neighbour resides
      Pair<Integer, Float> nn = nearest_neighbour((ArrayList<Float>) it.next(), W);
      it.previous();
      //This is the genotype for the nearest
      ArrayList<Float> nearest = W.get(nn.getFirst());
      //This is the codebook that matches the nearest neighbour
      ArrayList<Float> codebook = X.get(counter);
      //We want to map genotype to the set of codebooks
      //Map[ positions in W, positions in X ]
      ArrayList<Integer> tmp = new ArrayList<Integer>();
      if (newp.containsKey(nn.getFirst())) {
        tmp = newp.get(nn.getFirst());
      }
      tmp.add(counter);
      newp.put(nn.getFirst(), tmp);
      it.next();
      Float second = nn.getSecond().floatValue();
      q_err = q_err + second;
      ++counter;
    }
    //System.out.println("Done mapping " + System.currentTimeMillis() + " " + newp.size());
    //concern - scope?
    P = newp;
    orig.phenotype = P;
    orig.setQ_ERR(q_err);
    //return q_err;
    return orig;
  }
  /**
   * Method to find the median value in the codebooks
   * @param S The map of positions in codebook
   * @param W The resulting genotypes for each map position in the codebook
   * @param myVQ The codebook
   * @return The median values
   */
  public ArrayList<Float> calc_centroid(ArrayList<Integer> S, ArrayList<ArrayList<Float>> W, ArrayList<ArrayList<Float>> myVQ) {
    //myVQ is the codebook
    //Time to go back and look at all the mappings
    //Each S will point us to the set of W
    ListIterator s = S.listIterator();
    int scounter = 0;
    //ssize is the size of each internal vector< float > of W
    //size is the size of the vector< vector< float > >
    int ssize = S.size();
    int ssize = W.get(0).size();
    int size = S.size();
ArrayList<Float> w = new ArrayList();
//w.setSize(ssize);
w.ensureCapacity(ssize);
for (int j = 0; j < ssize; ++j) {
    w.add(0.0f);
}

while (s.hasNext()) {
    int locationInMyVQ = (Integer) s.next();
    //Vector<Float> tmp = W.get(scounter);
    ArrayList<Float> tmp = myVQ.get(locationInMyVQ);
    ++scounter;
    for (int i = 0; i < ssize; ++i) {
        Float tmpValue = tmp.get(i);
        if (tmpValue == null) {
            System.out.println("uh-oh " + (scounter - 1) + " - " + ssize + " - " + size);
        }
        Float oldValue = w.get(i);
        Float newValue = 0.0f;
        try {
            newValue = oldValue + tmpValue;
        } catch (Exception e) {
            System.out.println("Error " + tmpValue + " - " + oldValue + " - " + e);
        }
//w.setElementAt(newValue, i);
w.set(i, newValue);
    }
    for (short i = 0; i < ssize; ++i) {
//w.setElementAt(w.get(i) / size, i);
w.set(i, w.get(i)/size);
    }
    return w;
}

/**
 * Calculates the mean squared error
 * @param q_err
 * @return mse
 */
public float MSE(float q_err) {
    return q_err / (X.size() * K);
}

/**
 * Determines the Peak Signal-to-noise ratio
 * @param mse is the mean squared error
 * @return the PSNR given the mse
 */
public float PSNR(float mse) {
    // For monochromatic images, max2 equals 255^2,
    // where 255 is the maximum possible gray level
    // value in the input signal.
    // "mse" is the mean squared error
    double max2 = (dsup * dsup);
    return (float) (10 * Math.log10(max2 / mse));
//return -1.0 / (1 + mse);
}

/**
 * Local search method for the GA
 * @param P Phenotype
* @param W Genotype
* @param myVQ the codebook
* @return result of local search algo
*/
public ArrayList<ArrayList<Float>> update_codebook_by_kmeans(Map<Integer,
 ArrayList<Integer>> P, ArrayList<ArrayList<Float>> W, ArrayList<ArrayList<Float>> myVQ) {
    ArrayList<ArrayList<Float>> result = new ArrayList<ArrayList<Float>>();
    for (int counter = 0; counter < W.size(); ++counter) {
        result.add(counter, W.get(counter));
        if (P.containsKey(counter)) {
            ArrayList<Integer> tmp = P.get(counter);
            if (tmp.size() > 0) {
                ArrayList<Float> updatedCentroid = calc_centroid(tmp, W, myVQ);
                result.set(counter, updatedCentroid);
            }
        }
    }
    return result;
}

/**
 * Other method of the local search for this algo
 * @param s Scale factor
 * @param w Genotype
 * @param c centroid
 * @return result of local search
 */
public ArrayList<Float> update_codevector_by_lee(float s, ArrayList<Float> w,
 ArrayList<Float> c) {
    ArrayList<Float> tmp = new ArrayList<Float>();
    int size = w.size();
    tmp.ensureCapacity(size);
    for (int i = 0; i < size; ++i) {
        tmp.add(i, w.get(i) + s * (c.get(i) - w.get(i)));
    }
    return tmp;
}

/**
 * Local search method for GA
 * @param s Scale Factor
 * @param p
 * @param P Mapping of phenotype/genotype
 * @param W Genotype
 * @param myVQ Codebook
 * @param ourRandom Pseudo random number generator to use in this method
 * @return Result of local search
 */
public ArrayList<ArrayList<Float>> learning_by_lee(float s, float p, Map<Integer,
 ArrayList<Integer>> P, ArrayList<ArrayList<Float>> W, ArrayList<ArrayList<Float>> myVQ,
 RandomMTBMA ourRandom) {
    ArrayList<Float> updatedCodebook = new ArrayList<ArrayList<Float>>() {
        //P is a mapping of W (Individual.genotype) and myVQ.X
        //It says for each key in P (which is myVQ.X[P[x]], there are positions in W
        //Individual.genotype
        for (int i = 0; i < P.size(); ++i) {
            updatedCodebook = update_codebook_by_kmeans(P, W, myVQ);
        }
        else {
            ListIterator w = W.listIterator();
            for (int counter = 0; counter < W.size(); ++counter) {
                result.add(counter, W.get(counter));
                if (P.containsKey(counter)) {
                    ArrayList<Integer> tmp = P.get(counter);
                    if (tmp.size() > 0) {
                        ArrayList<Float> updatedCentroid = calc_centroid(tmp, W, myVQ);
                        result.set(counter, updatedCentroid);
                        //W[w] = calc_centroid( P.get( w ) );
                    }
                }
            }
            return result;
        }
    }
    return result;
}
int i = 0;

while (w.hasNext()) {
    int key = i;

    if (P.containsKey(key)) {
        ArrayList<Integer> tmp = P.get(key);
        if (tmp.size() > 0) {
            ArrayList<Float> c = calc_centroid(tmp, W, myVQ);
            float u = ourRandom.value_range(ourRandom.getSeed(), 0.0f, 1.0f);

            //if (u < p)
            ArrayList<Float> updatedCodevector = new ArrayList<Float>();
            updatedCodevector = update_codevector_by_lee(s, W.get(i), c);
            W.set(i, updatedCodevector);
            //else
            //*w = c;
            }
        }
    w.next();
    ++i;
    }

updatedCodebook = W;
}
return updatedCodebook;
}
A.13 – Changes to VisualPanel.java

/* VisualPanel.java
 * Terrain-Based Genetic Algorithm Visualization Project
 * Copyright 2003, 2004 Evolving Solutions.  All rights reserved.
 * Use is subject to license terms.
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VisualPanel extends JPanel implements ActionListener, ItemListener
{
    private Display display;
    private Renderer renderMethod;
    private Problem problem;

    private JCheckBox chkAutoRotate;
    private JCheckBox chkShowLabel;
    private JCheckBox chkShowScaleFactor;
    private JCheckBox chkShowPeak;
    private JCheckBox chkShowWeightAvg;
    private JCheckBox chkShowCity;

    private JComboBox cmbRenderMethod;
    private JTextArea txtaRenderDesc;

    private final static String rendererList[] = {
        "Wireframe", "Facet Shaded", "Flat Renderer" }
;
    /**
     * Initializes visualization panel to default state.
     */
    public VisualPanel(Display display, Renderer renderMethod, Problem theProblem )
    {
        this.display = display;
        this.renderMethod = renderMethod;
        this.problem = theProblem;

        Box hb;   // For Horizontal Boxes
        Box vb;   // For Vertical Boxes

        GridBagConstraints c;
        GridBagConstraints gb;

        JLabel l;
        JPanel p;

        this.setLayout(new BorderLayout());
        this.setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createTitledBorder("Visualization"),
            BorderFactory.createEmptyBorder(10, 10, 10, 10)));

        // Render Method
        vb = Box.createVerticalBox();
        vb.add(Box.createVerticalGlue());
hb = Box.createHorizontalBox();

l = new JLabel("Method: ");
l.setToolTipText("Select a method to render the visualization.");
cmbRenderMethod = new JComboBox(rendererList);
cmbRenderMethod.setMaximumSize(new Dimension(cmbRenderMethod.getMaximumSize().width,
cmbRenderMethod.getMinimumSize().height));
cmbRenderMethod.setToolTipText("Select a method to render the visualization.");
cmbRenderMethod.addActionListener(this);
l.setLabelFor(cmbRenderMethod);

hb.add(l);
hb.add(cmbRenderMethod);

vb.add(hb);
this.add(vb, BorderLayout.NORTH);

////////////////////////////////////////////////////////////////////////
// Description
////////////////////////////////////////////////////////////////////////
Font f = (Font) UIManager.get("Label.font");
txtaRenderDesc = new JTextArea(renderMethod.description());
txtaRenderDesc.setEditable(false);
txtaRenderDesc.setLineWrap(true);
txtaRenderDesc.setWrapStyleWord(true);
txtaRenderDesc.setBackground((Color) UIManager.get("Label.background"));
txtaRenderDesc.setForeground((Color) UIManager.get("Label.foreground"));
txtaRenderDesc.setFont(f.deriveFont((float) (f.getSize() * 9 / 10)));
this.add(txaRenderDesc, BorderLayout.CENTER);

////////////////////////////////////////////////////////////////////////
// Visualization Options
////////////////////////////////////////////////////////////////////////

vb = Box.createVerticalBox();

chkAutoRotate = new JCheckBox("Auto Rotate");
chkAutoRotate.setAlignmentX(Component.LEFT_ALIGNMENT);
chkAutoRotate.setToolTipText("Automatically rotate the visualization.");
chkAutoRotate.setSelected(false);
chkAutoRotate.addItemListener(this);

hb.add(chkAutoRotate);

chkShowLabel = new JCheckBox("Show Data Labels");
chkShowLabel.setAlignmentX(Component.LEFT_ALIGNMENT);
chkShowLabel.setToolTipText("Display terrain grid coordinate values.");
chkShowLabel.setSelected(false);
chkShowLabel.addItemListener(this);

hb.add(chkShowLabel);
hb.add(Box.createHorizontalGlue());

chkShowScaleFactor = new JCheckBox("Show Scale Factor");
chkShowScaleFactor.setAlignmentX(Component.LEFT_ALIGNMENT);
chkShowScaleFactor.setToolTipText("Display scale factor coordinate values instead of mutation/crossover.");
chkShowScaleFactor.setSelected(false);
chkShowScaleFactor.addItemListener(this);
hb.add(chkShowScaleFactor);

chkShowCity = new JCheckBox("Show City");
chkShowCity.setAlignmentX(Component.LEFT_ALIGNMENT);
chkShowCity.setToolTipText("Display the number of individuals in each cell.");
chkShowCity.setSelected(false);
chkShowCity.addItemListener(this);
hb.add(chkShowCity);

chkShowPeak = new JCheckBox("Show Highest Peak");
chkShowPeak.setAlignmentX(Component.LEFT_ALIGNMENT);
chkShowPeak.setToolTipText("Displays a point representing the " + "highest peak for the visualization.");
chkShowPeak.setSelected(false);
chkShowPeak.addItemListener(this);
hb.add(chkShowPeak);

chkShowWeightAvg = new JCheckBox("Show Weighted Average");
chkShowWeightAvg.setAlignmentX(Component.LEFT_ALIGNMENT);
chkShowWeightAvg.setToolTipText("Displays a point representing the " + "weighted average of solution locations for the visualization.");
chkShowWeightAvg.setSelected(false);
chkShowWeightAvg.addItemListener(this);
hb.add(chkShowWeightAvg);

vb.add(hb);
vb.add(Box.createVerticalGlue());
this.add(vb, BorderLayout.SOUTH);
if ( problem.getMemeticMode() )
{
    chkShowScaleFactor.setEnabled(true);
}
else
{
    chkShowScaleFactor.setEnabled(false);
}

/**
 * Handles User Events from <code>JComboBox</code> objects.
 */
public void actionPerformed(ActionEvent e)
{
    Object source = e.getSource();
    if ( source == cmbRenderMethod )
    {
        double scaleFactor = renderMethod.getScaleFactor();
    }
String name = (String) ((ComboBox) source).getSelectedltem();

if ( name.equals("Wireframe") )
{
    renderMethod = new WireFrame(display.getProblem());
} else if ( name.equals("Facet Shaded") )
{
    renderMethod = new HSRRenderer(display.getProblem());
} else if ( name.equals("Flat Renderer") )
{
    renderMethod = new FlatRender(display.getProblem());
}
txtaRenderDesc.setText(renderMethod.description());

if ( chkAutoRotate.isSelected() )
    renderMethod.setAutoRotate(true);
if ( chkShowLabel.isSelected() )
    renderMethod.setShowLabel(true);
if ( chkShowScaleFactor.isSelected() )
    renderMethod.setShowScaleFactor(true);
if ( chkShowPeak.isSelected() )
    renderMethod.setShowPeak(true);
if ( chkShowWeightAvg.isSelected() )
    renderMethod.setShowWeightAvg(true);
if ( chkShowCity.isSelected() )
    display setShowCity(true);
else
display setShowCity(false);

renderMethod.setScaleFactor(scaleFactor);
display setRenderer(renderMethod);

/**
 * Initializes Renderer to Display a New Problem.
 */

public void
initRenderer(Problem problem)
{
    String name = (String) cmbRenderMethod.getSelectedItem();

    if ( name.equals("Wireframe") )
    {
        renderMethod = new WireFrame(problem);
    } else if ( name.equals("Facet Shaded") )
    {
    }
    else if ( name.equals("Flat Renderer") )
    {
    }

txtaRenderDesc.setText(renderMethod.description());

if ( chkAutoRotate.isSelected() )
renderMethod.setAutoRotate(true);

if ( chkShowLabel.isSelected() )
    renderMethod.setShowLabel(true);

if ( chkShowScaleFactor.isSelected() )
    renderMethod.setShowScaleFactor(true);

if ( chkShowPeak.isSelected() )
    renderMethod.setShowPeak(true);

if ( chkShowWeightAvg.isSelected() )
    renderMethod.setShowWeightAvg(true);

if ( chkShowCity.isSelected() )
    display.setShowCity(true);

display.setRenderer(renderMethod);
}

/**
 * Handles User Events from <code>JCheckBox</code> objects.
 */

public void
itemStateChanged(ItemEvent e)
{
    Object source = e.getItemSelectable();

    if ( source == chkAutoRotate )
    {
        if ( e.getStateChange() == ItemEvent.SELECTED )
            renderMethod.setAutoRotate(true);
        else
            renderMethod.setAutoRotate(false);
    }
    else if ( source == chkShowLabel )
    {
        if ( e.getStateChange() == ItemEvent.SELECTED )
            renderMethod.setShowLabel(true);
        else
        {
            renderMethod.setShowLabel(false);
            renderMethod.setShowScaleFactor(false);
            chkShowScaleFactor.setSelected(false);
        }
    }
    else if ( source == chkShowScaleFactor )
    {
        if ( renderMethod.isShowLabel() )
        {
            if ( e.getStateChange() == ItemEvent.SELECTED )
                renderMethod.setShowScaleFactor(true);
            else
            {
                renderMethod.setShowScaleFactor(false);
                renderMethod.setShowScaleFactor(false);
                chkShowScaleFactor.setSelected(false);
            }
        }
        else
        {
            renderMethod.setShowScaleFactor(false);
            chkShowScaleFactor.setSelected(false);
        }
    }
    else if ( source == chkShowCity )
    
    else if ( source == chkShowWeightAvg )
if ( e.getStateChange() == ItemEvent.SELECTED )
   display.setShowCity(true);
else
   display.setShowCity(false);
} else if ( source == chkShowPeak )
{
   if ( e.getStateChange() == ItemEvent.SELECTED )
      renderMethod.setShowPeak(true);
   else
      renderMethod.setShowPeak(false);
} else if ( source == chkShowWeightAvg )
{
   if ( e.getStateChange() == ItemEvent.SELECTED )
      renderMethod.setShowWeightAvg(true);
   else
      renderMethod.setShowWeightAvg(false);
}

public void updateProblem( Problem newProblem )
{
   chkShowScaleFactor.setSelected(false);
   renderMethod.setShowScaleFactor(false);
   this.problem = newProblem;
   if( problem.getMemeticMode() )
   {
      chkShowScaleFactor.setEnabled(true);
   } else
   {
      chkShowScaleFactor.setEnabled(false);
   }
}
A.14 – Changes to WireFrame.java

/**
 * Displays the given data model on the specified <code>BufferedImage</code>
 * as a 3D wire frame terrain. If so configured, it may also draw labels
 * or show the highest or weighted average positions on the terrain.
 */

public void render(Point3D model[], BufferedImage canvas)
{
    boolean scaleModel = false;
    int w = canvas.getWidth();
    int h = canvas.getHeight();
    int sideLen = problem.getSideLength();

    String txt;
    Graphics2D g2 = canvas.createGraphics();
    g2.setBackground(Color.BLACK);
    g2.setColor(Color.GREEN);
    g2.clearRect(0, 0, w, h);
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);

    if ( isAutoRotate() )
        camOrt.setY((camOrt.getY() - 1) % 360);
    double c = Math.cos(-Math.toRadians(camOrt.getY()));
    double s = Math.sin(-Math.toRadians(camOrt.getY()));
    MTM.identity();
    MTM.set(0, 0, c);  // Rotate and Scale Model
    MTM.set(0, 2, -s);
    MTM.set(2, 0,  s);
    MTM.set(2, 2,  c);
    MTM.set(1, 1, getScaleFactor());
    MTM.postMultiply(VTM);

    // Transform to Screen Coordinates
    STM.set(0, 0, (double)  (w / 2));
    STM.set(1, 1, (double) -(h / 2));
    STM.set(3, 0, (double)  (w / 2));
    STM.set(3, 1, (double)  (h / 2));

    for ( int i = 0; i < sideLen; i++ )
    {
        for ( int j = 0; j < sideLen; j++ )
        {
            p = model[i * sideLen + j].transform(MTM);
            p = p.divide(p.getW());
            p = p.transform(STM);
            xHor[j] = (int) p.getX();
            yHor[j] = (int) p.getY();

            p = model[j * sideLen + i].transform(MTM);
            p = p.divide(p.getW());
            p = p.transform(STM);
            xVer[j] = (int) p.getX();
            yVer[j] = (int) p.getY();

            if ( yHor[j] < 10 || yVer[j] < 10 )
            { }
scaleModel = true;
}

// Draw Gridlines
g2.drawPolyline(xHor, yHor, sideLen);
g2.drawPolyline(xVer, yVer, sideLen);

if ( isShowLabel() )
{
  // Skip Labels for Large Graphs
  if ( sideLen > 30 && i % 2 > 0 ) continue;
  if ( sideLen > 50 && i % 3 > 0 ) continue;

g2.setColor(Color.YELLOW);
if( !problem.getMemeticMode() || !isShowScaleFactor() )
{
  // Draw Crossover Point Labels
  p = getCrossoverLabel(i, camOrt.getY()).transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

txt = Integer.toString(problem.getCrossoverPoints(i));
g2.setFont(fontNumber);
g2.drawString(txt, (int) p.getX(), (int) p.getY());

  p = getCrossoverLabel(sideLen / 2, camOrt.getY() + 180.0);
p = p.transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setFont(fontText);
g2.drawString("Crossover", (int) p.getX(), (int) p.getY());
  // Draw Mutation Rate Labels
  p = getMutationLabel(i, camOrt.getY()).transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

txt = dfmt.format(problem.getMutationRate(i));
g2.setFont(fontNumber);
g2.drawString(txt, (int) p.getX(), (int) p.getY());

  p = getMutationLabel(sideLen / 2, camOrt.getY() + 180.0);
p = p.transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setFont(fontText);
g2.drawString("Mutation", (int) p.getX(), (int) p.getY());
}
else
{
  // Show the scale factor instead of mutation/crossover
  // Draw Crossover Point Labels
  // Crossover is x
  p = getCrossoverLabel(i, camOrt.getY()).transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

txt = Float.toString(problem.getScaleFactor1Points(i*problem.getSideLength()));
g2.setFont(fontNumber);
g2.drawString(txt, (int) p.getX(), (int) p.getY());

p = getCrossoverLabel(sideLen / 2, camOrt.getY() + 180.0);
p = p.transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setFont(fontText);
g2.drawString("Scale1", (int) p.getX(), (int) p.getY());

// Draw Mutation Rate Labels
// Mutation is y - change the index
p = getMutationLabel(i, camOrt.getY()).transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

txt = Float.toString(problem.getScaleFactor2Points(i));
g2.setFont(fontNumber);
g2.drawString(txt, (int) p.getX(), (int) p.getY());

p = getMutationLabel(sideLen / 2, camOrt.getY() + 180.0);
p = p.transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setFont(fontText);
g2.drawString("Scale2", (int) p.getX(), (int) p.getY());

g2.setColor(Color.GREEN);
}
}
}

if ( isShowPeak() )
{
p = getPeakPoint().transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setColor(Color.WHITE);
g2.fillOval((int) p.getX() - 5, (int) p.getY() - 5, 10, 10);
}

if ( isShowWeightAvg() )
{
p = getWeightAvgPoint().transform(MTM);
p = p.divide(p.getW());
p = p.transform(STM);

g2.setColor(Color.CYAN);
g2.fillOval((int) p.getX() - 5, (int) p.getY() - 5, 10, 10);
}

if ( scaleModel ) setScaleFactor(getScaleFactor() * 0.9);
APPENDIX B
Software License

The Visual Terrain-Based Genetic Algorithm system is copyright 2003, 2004, 2006 by Evolving Solutions. All rights reserved. Usage is subject to the terms that comprise this appendix. The license itself is courtesy of Creative Commons, a non-profit organization. They may be reached at http://creativecommons.com. Aside from the creation of the license from an automated template system, Creative Commons is not a party to this license.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

"Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

"Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License.

"Licensor" means the individual or entity that offers the Work under the terms of this License.

"Original Author" means the individual or entity who created the Work.
"Work" means the copyrightable work of authorship offered under the terms of this License.

"You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
- to create and reproduce Derivative Works;
- to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
- to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform.
- You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement.

The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of
this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer
By offering the Work for public release under this License, Licensor represents and warrants that, to the best of Licensor's knowledge after reasonable inquiry:
Licensor has secured all rights in the Work necessary to grant the license rights hereunder and to permit the lawful exercise of the rights granted hereunder without You having any obligation to pay any royalties, compulsory license fees, residuals or any other payments; The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.

EXCEPT AS EXPRESSLY STATED IN THIS LICENSE OR OTHERWISE AGREED IN WRITING OR REQUIRED BY APPLICABLE LAW, THE WORK IS LICENSED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES REGARDING THE CONTENTS OR ACCURACY OF THE WORK.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, AND EXCEPT FOR DAMAGES ARISING FROM LIABILITY TO A THIRD PARTY RESULTING FROM BREACH OF THE WARRANTIES IN SECTION 5, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination
This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received
Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous
Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.


