GENETIC ALGORITHM AS A METHOD TO POSITION OBJECTS IN A 3 DIMENSIONAL SCENE

CHRIS VENTEICHER
1326 SIXTH STREET
HARVARD, IL 60033
V.TIGER@USA.NET

ABSTRACT

A genetic algorithm was used to position a graphical object in a three dimensional scene in an effort to reconstruct an image containing the object. Runs were performed with various parameter settings to test the system. The paper presents the results of these runs and conclusions that can be drawn from them. It was shown that the genetic algorithm could solve the problem for a simple test case. The paper describes the system, implications of the results and provides suggestions for future work.

INTRODUCTION

BACKGROUND, MOTIVATION, AND PURPOSE

The software described here uses a genetic algorithm to position a graphical object in a three dimensional scene. The genetic algorithm searches three-dimensional space to determine the position and orientation for the object. The goal is to place the object so that the scene looks as similar as possible to a know image. Both the object to place and the target image are provided beforehand. The genetic algorithm attempts to find the solution that reproduces the image.

Why is this important? Because determining where to put an object in a scene is a tedious error prone process. An efficient and competent automated method could increase productivity when generating computer animation. But perhaps more importantly an automated method could help enable significant new advances in video compression. A new video compression standard (MPEG 4) allows scenes to be communicated as a collection of graphical objects (ISO, 1998). The objects can be graphical models, textures, video or audio.

During compression raw video is broken into discrete objects so that each object is communicated independently. Decompression renders the objects to reproduce the original images. The key to achieving compression is avoiding the need to communicate objects over and over again for each frame. Rather, we want to communicate good objects once in a while and communicate information about where to place those objects to reproduce the video frames most of the time.

There are many aspects of this problem that a GA (genetic algorithm) could be applied to. For the software described in this paper the choice was made to focus on determining where an object should be placed in a scene to reproduce the original image. This is a critical part of the problem. Software was developed to act as a test platform to explore how a GA could be applied and to see if a GA would be effective on a problem of minimal difficulty.

OVERVIEW OF REMAINING SECTIONS

The remaining sections will cover related work, details of the implementation, and a discussion of extensions and ramifications. In order to convey details about what was done, the implementation section will explain the method for determining how similar the generated and target images are (fitness.) Information

about what parameters the GA adapts including the scope of the parameters and the patterns or building blocks (BB's) that were observed will be covered. Information about how the GA's parameters were configured to achieve the best results will be provided. Results of various GA runs will be presented followed by a summary of what has been accomplished.

RELATED WORK

Researchers have shown that a GA's can place graphical objects successfully. Caldwell and Johnston (Caldwell, 1991) showed that images of a face could be constructed by selecting and positioning a nose, mouth, eyes, forehead and chin. The GA used was able to assist in the placement of these parts of the face to produce images of a criminal suspect.

Caldwell and Johnston were able to show that the GA was able to productively converge on a solution. Each individual in the GA's population allowed 7 bits for each of the 5 objects for a total individual size of 35 bits. Of the 7 bits per object some of the bits determined what object (different noses etc) to use and the rest determined where the selected object would be positioned on the face. Even with the minimal number of bits per object Caldwell and Johnston specified that over 34 billion composite faces were possible.

The work was important because the GA was able to converge on a solution for a positioning problem in 2D space. In addition the discussion of the size and format of the parameters made clear the need to limit the scope and size of the individuals used to position 3 dimensional objects.

The fact that the GA in was able to converge to a solution in relatively short time provided a basis for believing the same might hold when positioning is extended to 3 space.

The ECGA package (Lobo, 1999) was selected as the GA. This was done primarily so linkage-learning (Harik, 1999) could be evaluated. Simple GA's tend to be susceptible to local maxima and tend to be impacted by extensive non-linearity in the search space (Thierens, 1993)(Goldberg 1989). Images by nature have distinct periods of non-linearity, local maxima and deceptive elements.

FITNESS: DO THE IMAGES LOOK THE SAME?

This is the question the GA must answer to determine fitness. The GA must rank all the individuals in its population by fitness to determine the makeup of the next generation. For each individual the GA must position and render an image file to compare against the target image.

If time were not an issue a human could compare the rendered image to the target image and assess fitness. This is the method used in to rank images of the criminal suspect in (Caldwell, 1991). But, time is an issue. The search space of possible positions for the object is large and a substantial number of individuals are required to search enough of the space to find a good solution. There simply is not enough time to have a human review each individual for each generation.

Because the GA must evaluate individuals quickly a computational assessment of fitness is made. As would be expected High fitness is obtained when the composted image looks very much like the original image. Low fitness is when the images differ significantly. After the test image is constructed the two images are compared on a pixel-by-pixel basis. The fitness is determined as the mean squared error of the two images (Teo, 1994).

```
for(pixel=0; pixel < number_pixels_in_image; pixel++){
   delta = image1[pixel].luminance - image2[pixel].luminance
```

```
    deltaSquaredSum = deltaSquaredSum + (delta * delta)
}
imageError = sqrt(deltaSquaredSum)
fitness = -1.0 * (imageError / number_of_pixels_in_image)
```

Inspection of the pseudo code will show that we are comparing the luminance (intensity) of the two images on a pixel-by-pixel basis. The red green and blue pixel components are summed to calculate luminance. But, it should be noted that color match is not part of the fitness calculation. Only intensity is considered in the fitness calculation at this time. Given the intensity of the corresponding pixels in both the rendered and target image the code proceeds to determine the pixel contribution as the difference between the pixel intensities squared. The contribution of each pixel is then summed with the contribution of all other pixels in the image. Total image error is calculated as the square root of the sum of the pixel contributions as would be expected to determine the mean squared error between two images.

Average error per pixel is calculated by dividing the total image error by the number of pixels in the image. Average error per pixel starts at 0.0 and gets larger as images differ more. Fitness is determined by multiplying average error per pixel by -1.0.

Fitness will have a maximum of 0.0 when the original and the constructed image are identical. Otherwise if the images are not identical fitness will be in increasingly large negative number proportional to the degree of difference. This is somewhat non-standard compared to existing objective functions in the ECGA package that operate with fitness values above zero. The choice was made to allow the fitness to be negative rather than shift the range up in value by an arbitrary amount. Care was taken to assure that the GA code handled the negative fitness values properly.

For the sum of squares method used the expected minimum fitness would be on the order of minus 10^3 if the images were completely different. However in the experiments conducted the fitness quickly converged to an asymptote between -2.0 and 0.0.

WHAT SPECIFIC STRUCTURE DOES THE GA ADAPT?

The GA maintains a population of individuals and is tasked with forming a series of new generations made up of offspring from the individuals in the previous generation. Before producing a new generation the GA assesses the fitness of the individuals in the current generation and gives the high fitness individuals more offspring in the next generation. The fitness determination is what was talked about in the previous section.

The test program created starts with a 3 dimensional object and a target image to place the object in. The program takes as input a VRML file containing the object. In fact the VRML file contains the target image as well.

To keep the project simple a single VRML file containing multiple objects was used. The program starts by rendering the entire VRML scene and storing the resulting image as the target image. The target image remains the benchmark for fitness and is unaltered from that time forward. The program proceeds by isolating one object in the VRML file to use as the test object. The test object is then under the control of the GA for all subsequent rendering.

Once the target image is established and the test object is identified the GA is instantiated. The GA reads a control file passed as an option when the program is started and creates a pre-specified number of individuals to evolve.

Each individual stores the information needed to place the test object in the scene. Two distinct pieces

of information are required: position and orientation. The position component determines the x, y, and z point in space where the object is located. The orientation component determines rotation about the x-axis, y-axis and z-axis and is used to point the object in the correct direction in space.

```
typedef struct {
    UINT16  rotX,rotY,rotZ,x,y,z;
} PositionOrientation;
```

Generation zero consists of individual's instantiated with random values for all parameters. The GA is free to begin processing. Fitness is associated with each individual by rendering the object as placed by the individual and calculating the difference from the original image. After obtaining fitness information for each individual the GA produces the next generation by applying selection and crossover to the population.

Provided a sufficient initial population size and proper choice of tournament size, and given the random initial values in the individuals, a sufficient volume of the search space is covered to allow the GA to converge on fit solutions.

SEARCH SPACE

How big is this problem? How much searching will the GA need to do to find the best solution? Search space is all possible distinct positions and orientations available within the scene. Each distinguishable position and orientation represents the most fundamental building blocks that must be considered.

The search space can be large but at the same time placing objects and evaluating fitness is a resource-limited task. The overhead associated with determining an individual's fitness is large because it takes time to render an image and calculate fitness. Care must be taken to limit the number of fitness evaluation. In other words, care must be taken to limit the search space that must be explored. The size of the individual was kept at a minimum to keep the search space as small as possible. Interesting parameters like scaling and texture mapping control were not included and parameters that were included did not allow excessive granularity.

The GA will tend to converge to local fitness maxima when the search space is not explored comprehensively. This was evident in the results when small population sizes and or high tournament sizes were used.

Even with efforts to keep the individual's small there is significant size and complexity. The search space can be enumerated by examining the structure mapped to the individual. Each individual represents one PostitionOrientation structure. The structure has 6 parameters and each parameter is 16 bits long. Floating point representation is not used for the parameters. Rather each parameter holds an unsigned 16-bit integer. Each parameter can be stored as 0, 65534, or any number in-between.

A parameter is not used as a large integer even though it is stored as an integer. Instead, the parameters are mapped to the range 0.0 to 1.0. This mapping is done by dividing the stored integer value by the maximum possible integer value of 65534. Hence, there are 65534 possible values for each parameter within the range 0.0 to 1.0.

Position is determined for each dimension as a percentage of the span of the VRML bounding box in that dimension. The VRML bounding box for the scene is computed at initialization and recorded for use later. For example if the bounding box spans from -5 to 5 in the X direction and the GA individual has a value of 0x3FFF for the X position parameter, the object would be located at position –1.0 on the X axis.

Object rotation does not require use of the bounding box information. The rotation parameters

are calculated as a value between 0.0 and 1.0 with a granularity of 1/65535 as well. However, rotation simply maps to a -180 to +180 degree spin on the axis represented by the parameter. A value of 0x7fff represents no spin.

Lets now consider the magnitude of the possible combinations given these parameters. As said previously each parameter in the individual is 16 bits long. There are X, Y, Z position parameters and an X-axis, Y-axis and Z-axis rotation parameters. Together the parameters total 96 bits. Even though we have attempted to minimize the complexity 96 bits allows a total 7.9 * 10^28 possible individuals.

With this said it should be noted that the search space would generally be much smaller. Here is why. The search space is only as big as the number of fundamental building blocks that must be considered. And the most fundamental building blocks in this problem are differences in position or orientation that can be perceived as visible changes in the intensity of pixels. If a change is so small it does not change the intensity of pixels in the image it is irrelevant. The nature and make up of the target object will play a large part in determining how much the search space is reduced. But, with realistic examples only positional shifts of some large fraction of a pixel size will result in intensity changes. The granularity for rotation would be even higher. Most objects have consistent shading across their surface and small changes in position or orientation would not be obvious. The real search space would be situation depended but generally much smaller than 10^{28th}.

BUILDING BLOCKS

Fit individuals don't just get copied to the next generation. Instead they are combined with other fit individuals to produce children having a combination of the traits of both parents. This section will analyze the traits, or building blocks, passed from generation to generation

We have already seen that an individual consists of two sets of parameters: position and orientation. With in each set there are 3 values; one value for each axis. The 6 parameter values in the structure are 16 bits each for a total of 96 bits per individual.

The BB's (building blocks) established during a GA run will be determined by the contents of the image and the object that is being positioned. However, it is possible to identify some of the high level relationships that are predicable and observable.

One of the most important observations was mention earlier in regard to perception of fitness. Changes in the parameters not resulting in changes in pixel intensities don't count. It is reasonable to assume that enough granularity is present that at least some of the low order bits for some parameters won't matter. Those bits that don't matter in determining fitness are not building blocks. The smallest building block is slightly larger than the bits that don't matter.

Now lets look at BB's from the other direction; the BB's that count the most. The winner is: position. Find the right combination of bits to place the object in the correct spot on the screen and you really have something. Why are we discounting orientation? Because in most cases orientation does not matter until you get the object in the correct space. Position does not matter as much if the object is a significant part of the image (in terms of size.) But even if the object is large, position needs to be close for orientation to matter in a consistent way.

The implication of this is that location must converge quicker than orientation. This is clearly backed up by observation. As tournament size is lowered and population is increased such that the GA successfully converges individuals will seem to "land" on the correct location before orientation converges. But if the GA converges too quickly an interesting phenomenon occurs. Orientation will converge before position as the BB for the incorrect local maxima takes hold.

Here is what happens. Away from the correct position the GA finds individuals are more fit if oriented to have minimum observable profile. The GA identifies this local maximum and locks on to it. By the time position converges orientation is already locked in. The end result has the object in the correct position but with the wrong orientation.

The true global maxima can only be found if orientation BB's take hold after position BB's. The GA must be properly configured for this too happen. Tournament size must be low and population size must be large. The nature of this problem is to have false, or local maximums. Care must be taken to assure that the global maximum is found.

GA OPERATORS

The ECGA package was chosen as the GA. The ECGA supports the standard GA operators of selection and crossover. The ECGA does not directly support mutation and therefore mutation was not considered in this work.

Tournament selection with out replacement was chosen as the selection method. The choice was made not to address issues unique to this task but because tournament selection has been shown to be a robust method in previous work (Goldberg, 1991). The attractive features of tournament selection are its tendency to converge quickly but without converging too fast at the beginning thus eliminating important diversity. Tournament selection was shown to avoid introducing noise associated with proportional scaling methods.

Single point crossover was used as it was directly supported by the ECGA. More complicated crossover mechanisms are possible but the linkage learning capabilities of the ECGA should provide the improvements that would be hoped for from a multipoint crossover approach.

Linkage learning capability was the basis for choosing the ECGA over a simple GA. As was mentioned earlier, simple GA's can be susceptible to local maxima and would be impacted by extensive non-linearity in the search space resulting from the complicated nature of images. The ECGA provides an opportunity to evaluate linkage-learning when determining if GA's are up to the object positioning task.

MAJOR PARAMETERS

It was expected that parameter selection would be crucial to this problem and this was quickly confirmed by the tests. An effort was made to understand the impact of various parameters and to find a combination that worked reliably. To this end a series of runs was performed to evaluate the combination of various population sizes and tournament sizes. Population sizes of 50,100, 200 and 300 individuals were evaluated. For each population size tournament sizes of 5, 10 and 20 were evaluated. The results can be seen in Figures 2, 3, 4 and 5.

Figure 3 shows tournament size = 10 with population size = 200 was the optimum parameter combination for this problem. Figure 4 shows how changes in tournament size impacts average fitness per generation. Figure 5 shows how changes in population size impacts average fitness per generation. In both cases the fixed parameter was chosen from the best case combination derived from Figure 3.

The effect of linkage learning in the GA was also analyzed for the each population size at tournament size 5. The results are presented in figure 1 as well. Linkage learning clearly provided an improvement.

A key parameter that remained fixed was probability of crossover. Probability of crossover remained fixed

at 1.0 such that no parents were directly copied to the next generation. Fit individuals seemed to take over such that the GA converged within 15 generations. There seemed to be no reason to decrease the probability of crossover thereby reducing the amount of searching that would be done within the 15 generations.

ANALYSIS OF OBSERVED RESULTS

The GA was able to place the object correctly provided the right combination of tournament size and population size was chosen. Figure 1 shows the results of a successful run captured at generation 15. In this image the target object was the desk. The GA was able to find the correct location at which to place the desk. The GA came very close to finding the correct orientation. And, the GA was able to do this at a rate of 0.2 seconds per individual fitness calculation.

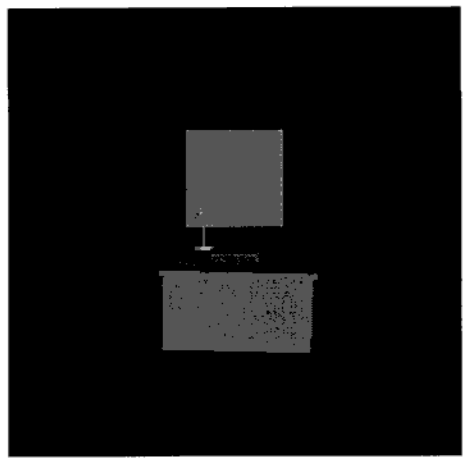

FIGURE 1

It was clear that linkage learning did improve the GA's performance. Runs with linkage learning disabled did not perform nearly as well as runs with linkage learning enabled. This can be seen in Figure 2 as well as the tournament size 5 no linkage-learning (5-noll) run in Figure 4. After establishing the value of linkage-learning all other testing was done with linkage learning enabled.

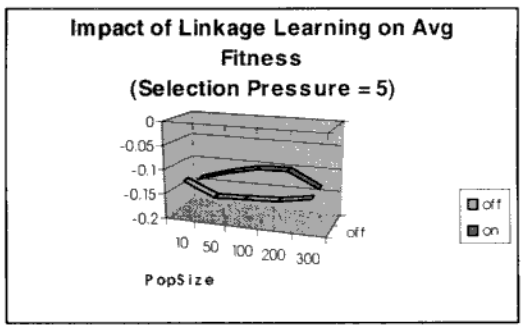

Figure 2

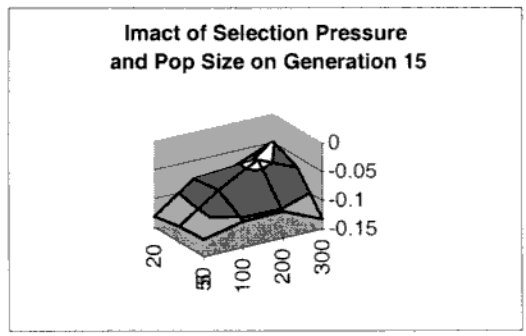

Figure 3

The data shows that individuals start out and maintain a consistently low fitness in the beginning. The performance is relatively the same regardless of what parameter choices were made.

Generation six marks the first point of departure in runs performed. Fitness starts to increase nonlinearly

after generation six in runs with tournament sizes of 10 and 20. Tournament size of 5 does begin to take hold eventually; but this does not happen until generation 12. Tournament size 5 with no linkage learning never takes hold and no significant jump in fitness is seen within 15 generations.

After the GA begins to converge the next milestone is when the GA runs out of diversity to exploit. For tournament size = 10 population sizes of 50 and 100 fitness growth stops by generation 9. After generation 9, fitness remained nearly constant for the two lowest population sizes. In fact the same thing is shown for population size 200 with tournament size 20. Diversity is key to growth and when diversity was used up and fitness growth stopped.

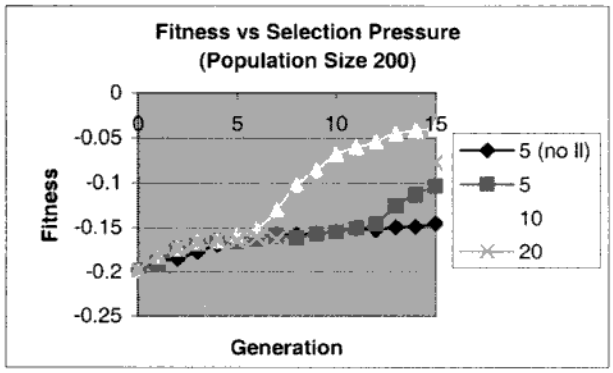

Figure 4

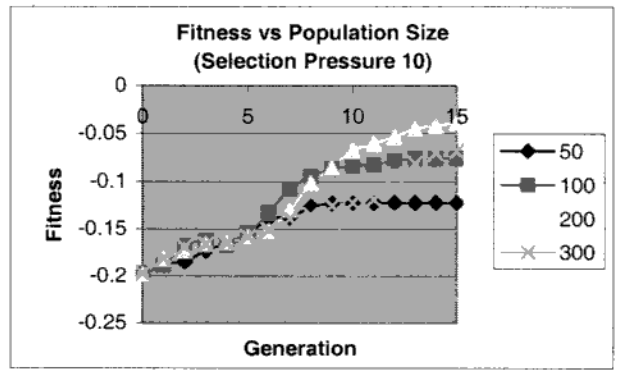

Figure 5

Tournament size of 10 with population size of 200 yielded the best fitness at generation 15. This is conveyed in Figure 4. However a close examination shows that population size 200 is reaching the peak of its curve and will not likely achieve much greater fitness. Population size 300 has consistently been less fit than population size 200 since generation 6. However, population size 300 has not reached its plateau yet and is still improving such that it may overtake population size 200 in a few more generations. The conclusion to be drawn from this is that 15 generations is appropriate for tournament size of 10 and population size of 200 but more generations are required to make higher population sizes valuable.

HARDWARE AND SOFTWARE USED

The ECGA platform was selected as the core GA for this project. The test program was constructed in C++ and operated on a SUN Ultra 60 workstation with 3 dimensional graphics capabilities. A public domain VRML library called CyberVRML was used to produce the images. The new work for this project linked the two packages through the data structure represented by the GA's individuals and provided the mechanism to evaluate the fitness of the result. The code is directly portable to other Unix or dos environments as all packages, libraries and custom components are portable. The graphics capabilities as used here are within the capability of most workstations and PC's with recent graphics cards.

PROBLEMS ENCOUNTERED AND EXTENSIONS ENVISIONED

A number of problems were encountered during development. Many of them were related to the implementation and were resolved simply by persistent and careful evaluation of results. But, some more

fundamental problems were observed as well.

For example, individuals had a strong pressure to minimize the profile of the object (by orienting it to occupy minimum screen space.) The GA then tended to converge to local maxima by placing the incorrectly oriented object. At the very least a good deal of rotational diversity is lost during the initial phases where the GA is finding the correct position for the object. Extensions should be made to hold off rotational convergence to allow position to converge ahead of rotation.

Another problem arises when the bounding box of the image is too large and the GA is free to position objects off of the screen. The GA quickly discovers objects rendered off screen produce better fitness than objects rendered on screen at incorrect locations. Objects rendered off screen represent a local maxia even though individuals are penalized for not improving the image. More robust solutions are needed to account for this issue.

EXTENSIONS AND RAMIFICATIONS:

RELEVANCE OF WORK TO CURRENT PRACTICE

The goal of this work was to provide initial proof of concept and to provide a platform for doing more work. Both of these goals have been achieved. It was shown that a GA could reposition a graphical object in a scene to match a target image. And a program that provides a platform for additional testing was created.

It should now be possible, using this program as a basis, to work with more complicated objects, images and algorithms. And it is reasonable to proceed because the GA has shown good performance with simple objects, images and algorithms. Success was obtained with fundamental tests and there is reason to believe more complicated applications could be made to succeed as well.

IMPORTANT EXTENSIONS

An initial important extension would be to make the fitness function respond with a fitness that more accurately models human perception (Teo, 1994). This would include assessment of color as well as luminance.

And, a method of overseeing and managing the GA will be required. It seems clear that a single set of parameters will not be sufficient for all scenarios. Many images will be solvable in a consistent amount of time. But some images will require much more processing to solve. It will be important to have an intelligent way of choosing population size and tournament size that works well on any image. This is a resource-limited task and it will be important for the GA to find solutions as quickly as possible.

SUMMARY AND CONCLUSIONS

The goal of this work was to provide initial proof of concept and to provide a platform for doing more work in the future. Both of these goals have been achieved. It was shown that a GA could reposition a graphical object in a scene to match a target image. And a program that provides a platform for additional testing was created.

The figures presented above show that the GA, with the appropriate settings is capable of identifying the correct building blocks and converging on a good solution for the 3 dimensional object positioning problem. The accompanying images show a solution that is a near perfect fit to the original image. And, the GA accomplished the solution shown in reasonable time.

The presented in the preceding graphs shows parameter choices are critical to finding a good solution to the positioning problem. These parameters are not the correct ones for all images and objects. But the need to manage the choice of parameter values for success has been identified. The results shown may be useful as a starting point for parameter choices for other images and objects. However an important extension would be to establish a computational way of determining these values.

It will be interesting to see if the success achieved to date can be generalized to more complicated images.

REFERENCES

ISO/IEC. (1998). International Standard 14496-1 (MPEG 4-Generic Coding of Audio-Visual Objects).

Caldwell, C. and Johnston, V.S. (1991) Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm, Proceedings of the fourth international conference on Genetic Algorithms, pp. 416-421.

Thierens, D. and Goldberg, D.E. (1993). Mixing in genetic algorithms. Proceedings of the Fifth International Conference on Genetic Algorithms pp. 38-45.

Goldberg, D.E., Korb, B. and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. Complex Systems, vol. 3, no. 5, pp. 493-530

Lobo, F.G. and Harik (1999). G.R. Extended Compact Genetic Algorithm in C++, ILLiGAL Technical Report 99016, 1999.

Harik, G.R. (1999). Linkage Learning via Probabilistic Modeling in the ECGA, ILLiGAL Technical Report 99010.

Teo, P. and Heeger, D. (1994). Perceptual Image Distortion, First IEEE International Conference on Image Processing, vol 2, pp. 982-986.

Goldberg, D.E. and Deb, K. A (1991). Comparative Analysis of Selection Schemes Used in Genetic Algorithms, Foundations of Genetic Algorithms, pp. 69-93.