# Evolving Boids: Using a Genetic Algorithm to Develop Boid Behaviors

**Jamison F. Conley**

Department of Geography, Pennsylvania State University,
University Park, PA, 16801
Tel: +1 814-865-1666
Email: jfc173@psu.edu

**Abstract**

The detection and analysis of clusters has become commonplace within geographic information science and has been applied in epidemiology, crime prevention, ecology, demography and other fields. One of the many methods for detecting and analyzing these clusters involves searching the dataset with a flock of boids (bird objects). While boids are effective at searching the dataset once their behaviors are properly configured, it can be difficult to find the proper configuration. Since genetic algorithms have been successfully used to configure neural networks, they may also be useful for configuring parameters guiding boid behaviors. In this paper, we develop a genetic algorithm to evolve the ideal boid behaviors. Preliminary results indicate that, even though the genetic algorithm does not return the same configuration each time, it does converge on configurations that improve over the parameters used when boids were initially proposed for geographic cluster detection. Also, once configured, the boids perform as well as other cluster detection methods. Continued work with this system could determine which parameters have a greater effect on the results of the boid system and could also discover rules for configuring a flock of boids directly from properties of the dataset, such as point density, rather than requiring the time-consuming process of optimizing the parameters for each new dataset.

## 1. Introduction

The detection and analysis of clusters has become commonplace within geographic information science and has been applied in epidemiology (e.g. Timander and McLafferty, 1998), crime prevention (e.g. Anselin *et al*., 2000), ecology (e.g. McKenna, 2003), and demographics (e.g. Dorling, 1998). Many methods for detecting and analyzing clusters have been proposed and analyzed, ranging from statistical approaches (e.g. Mantel, 1967; Kulldorf, 1997) to machine learning heuristics (e.g. Duczmal and Assunção, 2004). One promising approach is Macgill and Openshaw's (1998) use of a flock of boids to find clusters, especially in its generation of a results surface that can be used to generate the probability that a point is in a cluster instead of a distinctly-bounded set of clusters. However, finding good parameters for the agent behaviors was difficult. Genetic algorithms have been successfully used to configure neural networks (Deb *et al*.,

2002), so they may also be useful for configuring artificial life flocks. This paper begins work on using a genetic algorithm to configure an artificial life flock.

The remainder of the introduction provides background information on artificial life and genetic algorithms. Section 2 discusses prior work in this area in greater detail. Sections 3 and 4 discuss the development of the genetic algorithm; section 3 focuses on the theoretical aspects and section 4 focuses on our implementation. We present results from initial tests of the system in section 5 and conclude in section 6.

## 1.1 Artificial life background

Artificial life methods use agents to mimic the behavior of living creatures to solve computational problems. A common technique is to have flocks of agents, called boids (bird objects) move through a geographic and/or data space. Macgill and Openshaw (1998) developed a program to use flocks of boids to find geographic clusters. However, it was difficult to configure the parameters controlling the boids' behaviors. Vande Moere (2004) used boids to visualize patterns in a database. Each boid in Vande Moere's flock represents a single entry in the database, rather than a separate agent that scans the database. While cluster detection is not a focus of Vande Moere's flock, it is possible to see subgroups of boids within the flock behaving differently from the rest of the flock and interpret this as a cluster. This research develops a genetic algorithm to evolve the behaviors of Macgill and Openshaw's boids as the boids search a multidimensional geographic space for data clusters.

## 1.2 Genetic algorithm background

Genetic algorithms are a useful method for parameter optimization (Deb *et al.* 2002). In the context of parameter optimization, a genetic algorithm needs two pieces: a genetic data structure and a fitness function. A data structure is genetic if it supports the evolutionary operations of crossover and mutation, diagrammed in Figures 1 and 2. A commonly-used structure is a fixed-length array of real numbers. The crossover operation takes two instantiations of the data structure (parent genes), mixes parts of the data structure, and produces one or more child genes that are a combination of the two parent genes. With an array of real numbers, the child could have some of its array elements from one parent and the remainder from the other, as in the top of Figure 1, or its array elements could be the average of those elements in the parent genes, as in the bottom of Figure 1. The mutation operation (Figure 2) takes one gene and makes a small change, usually slightly incrementing or decrementing a single value within the gene, such as decreasing the fourth value from 12 to 11.

The fitness function takes a gene and measures how well it solves a particular problem. All fitness functions are problem-specific. In this research, the problem is how well the parameters in the array configure the artificial life program. The development of the fitness function is discussed in greater detail in section 3.2.

The genetic algorithm starts with a population of genes, then iteratively uses the scores of the fitness function to select some of the better fit genes to survive into the next generation and, using the crossover and mutation operations, produce children to fill the
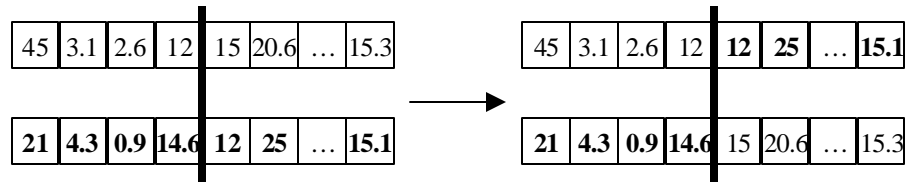
Some parameters from each parent
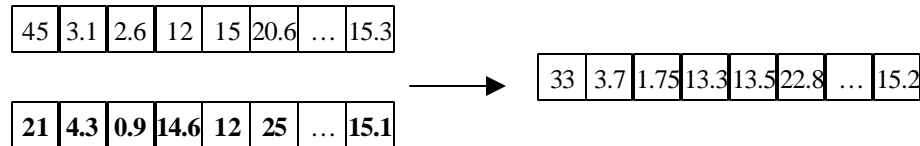


Average of the two parents

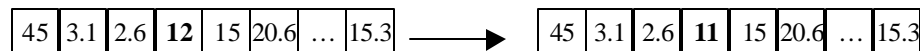Figure 1. Two methods of crossover with arrays of real-valued numbers.



Figure 2. A method of mutation for an array of real-valued numbers.

remainder of the next generation. This process repeats either for a fixed number of generations or until the genes converge onto identical or nearly identical configurations.

### 1.3 Goals of this work

By using a genetic algorithm to configure the artificial life program, we hope to find a set of parameters that improves upon the parameters estimated by Macgill and Openshaw (1998) when they originally developed the artificial life program.

This research will address two issues:

1) How will the effectiveness of a parameter configuration be determined? The genetic algorithm needs a fitness function to distinguish between configurations that work well and those that work poorly; how should a measurement of a configuration's effectiveness be taken, understanding that a configuration can have different rates of success on different datasets?
2) How does the hybrid genetic algorithm/artificial life approach compare against other machine learning techniques for cluster detection? Under what situations will the hybrid approach outperform other techniques, and vice versa?

## 2. Previous Work

### 2.1 ALife in cluster detection: geoboids

Computational agents have been used to model living systems since at least 1987, when Reynolds presented a computer program to simulate the flight of each bird within a flock. Macgill and Openshaw (1998) used this idea to drive a search through a spatio-temporal dataset to find clusters of points. The geoboids that they developed follow the Reynolds paradigm with a few modifications specific to cluster detection. The primary change is the addition of an attractiveness value for each boid. In Reynolds' flock, each boid was equally attracted to the other nearby boids in the flock. In Macgill and Openshaw's flock, though, boids flock more strongly towards other boids with a high attractiveness and away from those with very low attractiveness values. Because the density of cases within a boid determines its attractiveness value, the flock is able to wander its way away from areas with few or no cases to areas with a higher density of cases, and thus, areas with more clusters.

### 2.2 Difficulties in running geoboids

While artificial life techniques such as boids are effective at searching a dataset once they are properly parameterized, they are difficult to parameterize due to the large numbers of parameters that can affect a boid's behavior and the complex interactions between these parameters (Macgill and Openshaw, 1998; Agar, 2003; Vande Moere, 2004). Macgill and Openshaw simply estimated parameter values that made sense, although no tests were done to find optimal parameter configurations because of the lengthy running time of each boid run (on a late 1990s model computer) and because the large number of parameters (at least 20) explodes the amount of processing time needed to search the parameter space well beyond what a standard computer from 1998 could handle. With modern computers, though, such a search becomes possible.

### 2.3 GAs in parameter optimization

Genetic algorithms have been developed specifically to search through a real-valued parameter space (Deb *et al*., 2002). They have been used to develop a topology for neural networks (Ragg, 2002), and to parameterize neural network models for estimating the strength of particleboard (Cook *et al*., 2000) and forecasting electrical loads (Satpathy, 2003). In these applications, genetic algorithms have been proven useful for configuring other heuristic search methods, primarily neural networks. Their utility springs mainly from their ability to escape local optima and continue searching for a global optimum, even working towards solutions that may not be near the starting location for the population (Deb *et al*., 2002).

## 3. Approaches to Solving the Problem

### 3.1 Why use a GA to parameterize ALife?

Previous work with geoboids used trial and error approaches to configure the artificial life parameters (Macgill and Openshaw, 1998). As described above, genetic algorithms have been used to successfully configure the topology and parameters of a neural

network. Having proven useful in configuring one machine learning technique, genetic algorithms might also be useful for configuring the size and behaviors of a population of artificial life agents.

We identified twenty parameters of the artificial life system that are optimized by the genetic algorithm. These are named and described in Table 1.

### 3.2 How to measure the fitness of an artificial life configuration
The largest challenge in developing a genetic algorithm for configuring other systems is in determining the fitness of a particular configuration. In cases where the number, size, and distribution of clusters are known beforehand, this is not a major issue; if running the artificial life system with that configuration found all clusters, without including false positives, in a short amount of time, then the configuration is fit. This is the idea behind the first method, which is given in section 3.2.1. However, if the information about the clusters is not known beforehand, there are some ad-hoc rules of thumb that we propose in section 3.2.2.

### 3.2.1 Compare with known results
To compare the results of an artificial life geographical cluster detection run with known clusters given by another system, we propose turning both sets of results into binary black and white results surfaces. If a point in the geographic space is in a cluster, then it is colored black. Otherwise, it is colored white. Errors, both false positives and false negatives, will be different colors, while areas that are correctly determined to be inside or outside a cluster will be the same color. Therefore, the accuracy of the artificial life configuration can be determined by finding the percent of the area that is the same color in both surfaces.

This method works well for geographic clusters, but spatio-temporal clusters or space-attribute clusters could still be misclassified. For example, if a spatio-temporal cluster is in Pennsylvania during January, but the artificial life system detects a cluster in Pennsylvania in June, it will be an incorrect result, although it would be counted as correct by the comparison method just given. Adding a check to measure agreement for temporal and data attributes overcomes this problem.

### 3.2.2 Ad-hoc measures of system performance
When the correct results are not known beforehand, the method of section 3.2.1 cannot be applied, and running another method to find the clusters so that the artificial life system can be correctly configured to find the clusters defeats the purpose of running the geoboids in the first place. Therefore, for situations when the cluster information is not already known, we propose the following five ad-hoc measurements.

First, if configuration A returns more distinct results than configuration B, configuration A is better. This relies on distinguishing between new results and duplicate results. If a result is found that substantially overlaps another boid in the geographic domain (> 50% of the area of both boids) and also in the temporal and attribute domains, then it is considered to be a duplicate of the earlier result. If it does not have this overlap, then it is

Table 1.  Artificial life parameter names and descriptions.

| Parameter name | Parameter description |
| --- | --- |
| Number of boids | The number of boids in the population. |
| Panic radius | The radius of the circle a boid looks in to determine whether or not it is in an area crowded with boids. |
| No. of neighbors for panic | The number of other boids needed to be within a circle of radius *panic_radius* to make it crowded. |
| Magnitude of wander vector | The size of the wander vector: higher values make the boids' movement more independent while low values make them flock together more quickly. |
| Attraction for good boids | How much a boid flocks with another boid is based on the other boid's performance.  If the other boid is good, then a boid will be attracted to it by this value. |
| Attraction for neutral boids | Attraction value for boids that are performing OK. Less than the attraction value for good boids. |
| Attraction for bad boids | Attraction value for boids that are performing poorly. This is often negative, which means it repels other boids. |
| Good-neutral border | The significance value (which is calculated using a Poisson distribution) that determines whether or not a boid is performing well for the attraction values. |
| Bad-neutral border | Same as good-neutral border, but for determining the bad boids instead. |
| Max velocity for good boids | The maximum velocity for boids with a significance value above the good-neutral border. |
| Max velocity for neutral boids | The maximum velocity for boids with a significance value between the good-neutral border and the bad-neutral border. |
| Max velocity for bad boids | The maximum velocity for boids with a significance value below the bad-neutral border. |
| Lookout for very good boids | Each boid adjusts its speed and direction by finding all boids within a lookout distance and, using their significance values, flocking to (or away from) those other boids.  This is the lookout for very good boids. |
| Lookout for good boids | The lookout distance for good boids, where good is using the good-neutral cutoff below, not the good-neutral border above. |
| Lookout for neutral boids | The lookout distance for neutral boids, using the cutoffs below, not the borders above. |
| Lookout for bad boids | The lookout distance for bad boids, using the cutoffs below, not the borders above. |
| Very good-good cutoff | Cutoff between very good and good lookout distances. |
| Good-neutral cutoff | Cutoff between good and neutral lookout distances. |
| Neutral-bad cutoff | Cutoff between neutral and bad lookout distances. |
| Max change in velocity | The maximum change a boid can make in its velocity in a single time tick. |

a distinct, new result. If this distinction were not made, then a configuration that constantly re-discovers the same cluster could appear better, even though it has really only found one cluster. Also, this works only because the size of the boids is not a parameter included in the optimization; if the boids' size could be altered, then many small clusters would be preferred to one large cluster covering the same area. This can be overcome by finding the area of the black regions in the result surface described in section 3.2.1 instead of simply counting the number of distinct results. The theory behind this measurement is that a configuration that finds more clusters is preferable to a configuration that finds fewer clusters. In this particular system, it is unlikely that large numbers of false positive would skew this measurement because a boid only returns a result if it has a Poisson significance of $> 0.999$. Since there are approximately 500 evaluations made in each run (see section 4 for details), the number of false positives in each run will likely be 0 or 1 (out of approximately 50 distinct results), and thus will probably not likely have a large impact on this measurement.

Second, if configuration A returns a lower percentage of duplicate results than configuration B, configuration A is preferred. If there are many duplicate results from a configuration, then the system is wasting time finding the same cluster multiple times. A configuration that does not waste this time is better than one that does.

Third, boids that end up in empty regions of the dataset die. A configuration that produces a large number of dead boids is spending too much time searching empty areas. Therefore, if configuration A produces fewer dead boids than configuration B, configuration A is better.

Fourth, if two configurations find roughly the same results, but configuration A uses fewer evaluations than configuration B uses, configuration A is better. As datasets increase in size and dimensionality, scalable methods are needed. Because it found the same results in less time (assuming one evaluation always takes the same amount of time), configuration A is more scalable than configuration B.

Lastly, again, if the results are roughly the same, but configuration A searched more of the dataset than configuration B, then configuration A is better. We can assume that, since its search was more thorough, configuration A is less likely to miss a cluster than configuration B. This is not guaranteed to be the case, as a large boid could pass over and evaluate the area of a small cluster, but because the boid was too big, not see the cluster. Even so, a configuration that searches more thoroughly is preferable.

## 4. The system we implemented

### 4.1 Where the boids came from
We use the geoboid system developed by Macgill and Openshaw (1998), which uses circular boids in an artificial life flock to find the spatial location of clusters. Within each of the agents is a small genetic algorithm that searches to find the temporal and data attribute bounds that maximize the significance of the cluster at that spatial location. The

parameters that we optimize over affect only the flocking behavior, not the internal genetic algorithm. Section 2.1 describes the flocking mechanism used by the geoboid system.

**4.2 How we combined the various measures of fitness**
We combined the various ad hoc measures from section 3.2.2 into a method that determines whether one configuration is better fit than another. First, the method runs the two configurations for 500 evaluations each. We use a constant number of evaluations rather than a constant number of ticks because a constant number of ticks would bias towards large flocks, since large flocks can search more of the dataset than small flocks in a constant number of ticks. Also, keeping the number of evaluations constant approximately keeps the running time constant, which is more useful since users are more likely to be interested in minimizing the running time rather than the number of iterations through the flock. We also collect the following statistics about the run: number of total results, number of distinct results, number of dead boids, the tick number on which each distinct result was discovered, and the number of times each point in the dataset was examined. Only running each configuration once is risky because the results from different runs with the same configuration can vary greatly. However, while running each configuration multiple times would make the fitness measurement more reliable, it would unacceptably increase the running time (see section 5.2.2). Also, while a poor configuration might occasionally outperform a good one, only good configurations are likely to perform consistently well over twenty generations. Therefore, the problem is limited to the chance that a good configuration might be accidentally eliminated before the genetic algorithm can converge on it, rather than letting a poor configuration survive among good ones.

The method sequentially applies the five ad hoc measures given in section 3.2.2. If a measure determines one configuration to be better, then the method halts, returns the better configuration, and does not use the other measures. Therefore, we apply them such that the one we consider to be most important is first, and the least important is last.

We initially compare the number of distinct results found by each of the configurations. If one configuration has at least five more distinct results than the other, then the configuration with more results is returned as better fit. Next, each configuration's ratio of distinct results to total results is compared with the other. If one configuration has at least five percent more distinct results than the other, then it is returned as better fit. Thirdly, the number of dead boids resulting from each configuration is compared. If one configuration has at least fifty more dead boids than the other, the configuration with fewer dead boids is returned as better fit. Next, the method orders the tick numbers for the distinct results, and finds the median tick number. Then it converts this to the median number of evaluations by multiplying the tick number by the number of boids in the flock. Like running the flock for a constant number of evaluations in stead of a constant number of ticks, this eliminates bias toward large flock sizes that would be introduced by simply using the median tick number. If one configuration took 25 or more evaluations longer than the other, then the configuration that used fewer evaluations to find the clusters is returned as better fit. Finally, if none of the previous measures are able to

distinguish between the two configurations, we compare the number of points that were never examined and return the configuration with fewer unexamined points as better fit.

Once we implement it, we will use the surface comparison method described in section 3.2.1 to evaluate whether or not the ad hoc methods are accurate measurements of configuration fitness. For this comparison, we will use cluster surfaces generated by the geoboid system. An example surface is seen in Figure 3.

The surface will be converted into a two-tone image, where every pixel inside a cluster is black and every pixel that is not in a cluster is white. To compare two surfaces, we will perform a bitwise comparison on each pixel. We can then calculate the percent agreement between the two images and use this to determine how fit the configuration is. A greater agreement with the known clusters will indicate a better fit configuration.
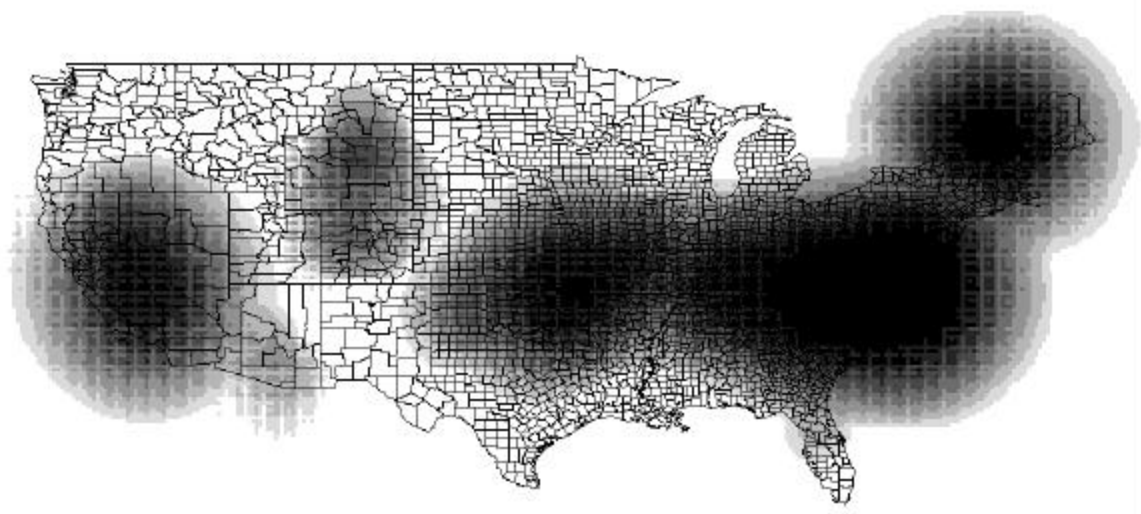
Figure 3. A sample results surface. This is the surface generated by GAM.

### 4.3 The final algorithm
The genetic algorithm we use to breed and evolve the configurations has a population of twenty configurations and halts after twenty generations. The chromosome that it uses is an array of the real-valued numbers for each of the parameters listed in Table 1. Survival into the next generation and selection of parents for crossover both use a 2-tournament selection method. Two configurations are randomly chosen, and the better fit configuration, as determined by a method from section 4.2, survives into the next generation or is used as a parent for crossover.

The crossover function takes two parent configurations, generates $r$, a random number between 0 and 1, and calculates the value of each parameter in the child configuration according to equation (1).

$$child.param = parentOne.param + (parentTwo.param - parentOne.param)*r \qquad (1)$$

The mutation function selects a random parameter, generates a random number between 0 and five percent of that parameter's current value, and either adds or subtracts this random number to the parameter's current value.

The final algorithm is as follows.

1) Initialize the population with 20 random configurations.
2) For twenty generations, do lines 3-18
3)      Run each configuration and collect statistics about the run.
4)      For ten iterations, do lines 5-7.
5)          Select two configurations at random.
6)          Compare them using a fitness function from section 4.2.
7)          Place the better configuration in the next generation.
8)      For ten iterations, do lines 9-16.
9)          Select two configurations at random.
10)         Compare them using a fitness function from section 4.2.
11)         Use the better configuration as a parent for crossover.
12)         Select two configurations at random.
13)         Compare them using a fitness function from section 4.2.
14)         Use the better configuration as the other parent for crossover.
15)         Perform crossover on the two parents.
16)         Place the child in the next generation.
17)      Select a configuration at random from the next generation and mutate it.
18)      Replace the population with the next generation.
19) Return the population.

## 5. Testing and Results

We trained the genetic algorithm on a simulated dataset based on the United States at a county level of detail. The dataset is similar to the distribution of West Nile Virus throughout the United States, although it introduces clusters by locating most of the cases in narrow bands across the country. In Figures 9 through 11, the counties with a West Nile Virus case are represented as darker dots.

The results we received were not quite as we had anticipated. While the genetic algorithm did converge each time we ran it, it did not always converge on the same parameter settings. While some parameters did not vary much from run to run, some varied quite a bit. This could be indicative of one of three issues: the genetic algorithm is not correctly set up; the parameter optimization space has many local optima, or the parameters that vary greatly have little impact on the results. At this point in time, we cannot determine which of these issues is operating.

First, the genetic algorithm used for parameter optimization may itself be configured poorly and thus unable to escape local optima. The best way to overcome this would be to increase the population size because that would introduce more genetic material in the

initial configurations, and give the algorithm a better chance of finding the optimal geoboid configuration. However, because the current genetic algorithm already takes a long time to run (see section 5.2.2), increasing the population size could make it prohibitively time-consuming to run.

Second, the parameter space for the artificial life flock may have so many local optima that even an optimization method as capable of escaping local optima as a genetic algorithm cannot guarantee it will converge on the global optimum. Similarly, the parameter space could have many optima of nearly the same fitness, thus having many optima that are almost as good as the global optimum. In this case, the genetic algorithm may simply be converging on a different global optimum or near-optimum each time.

Finally, there have not been systematic tests to determine how much each of these parameters actually affects the outcome of the artificial life run. Therefore, the parameters that vary wildly may simply have a minimal effect on the outcome. If this is true, then the genetic algorithm will simply optimize on the other parameters and the genetic algorithm will not always converge on the same value for the unimportant parameters.

## 5.1 Did it improve upon the parameters used in Macgill & Openshaw (1998)?
As mentioned above, the parameters that the algorithm converged on varied from run to run. Table 2 shows the original configuration used by Macgill & Openshaw and the configurations from three runs of the genetic algorithm. For the genetic algorithm runs, the values given are the mean value for all the configurations in the final generation.

Some constant patterns emerge from Table 2. First, the number of boids should be smaller than Macgill and Openshaw estimated. Also, the density of boids before claustrophobia causes boids to relocate should be higher. While the panic radius and the number of neighbors needed to induce panic vary, the density of neighbors within that radius varies less. The value of $panic\_neighbors / panic\_radius^2$ is between 8 and 12 for all three genetic algorithm runs. Because the radius of most boids is between 2 and 4, the maximum velocities are much larger with the genetic algorithm runs. Finally, the lookout radii for finding other boids to flock with should generally be larger than Macgill and Openshaw estimated (1998). However, many of the parameters, such as the magnitude of the wander vector and the maximum velocities, vary between runs.

To determine the effectiveness of each of these configurations, they were run for one minute on the same dataset on which the genetic algorithm trained. The cluster surfaces of each of these runs and GAM are shown in Figures 4 through 8. As a comparison between the surfaces generated from boid runs and the GAM surface shows, the configurations developed by the genetic algorithm are an improvement over the parameters from Macgill and Openshaw. Most of this improvement is in the central United States, where GAM and the genetic algorithm configurations include clusters, but the Macgill and Openshaw parameters do not find them. However, there are also a few clusters in the north-central part of the country that have clusters for the genetic algorithm runs but not for GAM.

Table 2.  Configurations from the genetic algorithm and from Macgill & Openshaw (1998).

| Parameter name | Macgill & Openshaw | GA Run 1 | GA Run 2 | GA Run 3 |
|---|---|---|---|---|
| Number of boids | 20 | 12.05 | 11.95 | 16.8 |
| No. of neighbors for panic | 10 | 8.75 | 4.3 | 9.95 |
| Panic radius | 1.5 | 0.974628 | 0.600754 | 1.070286 |
| Magnitude of wander vector | 0.2 | 0.150693 | 0.18272 | 0.484051 |
| Attraction for good boids | 0.8 | 0.819393 | 0.810729 | 1.258894 |
| Attraction for neutral boids | 0 | -0.0894 | -0.10287 | 0.370662 |
| Attraction for bad boids | -0.4 | -0.36375 | -0.80191 | -0.4231 |
| Good-neutral border | 0.7 | 0.825753 | 0.718952 | 0.771115 |
| Bad-neutral border | 0.1 | 0.482128 | 0.193679 | 0.280579 |
| Max velocity for bad boids | 1 * radius | 40.42976 | 38.4078 | 25.04387 |
| Max vel. for neutral boids | 0.75 * radius | 35.48538 | 34.19189 | 32.33065 |
| Max velocity for good boids | 0.5 * radius | 23.20348 | 14.53747 | 22.82742 |
| Very good-good cutoff | 0.9 | 0.918737 | 0.918365 | 0.909699 |
| Good-neutral cutoff | 0.7 | 0.605975 | 0.701519 | 0.597676 |
| Neutral-bad cutoff | 0 | 0.388308 | 0.049073 | -0.08859 |
| Lookout for very good boids | 1 | 5.722275 | 7.117242 | 6.926872 |
| Lookout for good boids | 2 | 5.087923 | 5.81815 | 4.767821 |
| Lookout for neutral boids | 3 | 3.316885 | 3.325939 | 6.346005 |
| Lookout for bad boids | 5 | 8.326031 | 3.524323 | 3.090013 |
| Max change in velocity | 0.8 | 0.723079 | 0.459964 | 0.818735 |



Figure 4.  Macgill & Openshaw.
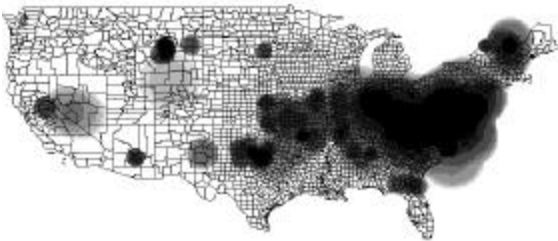


Figure 5.  GA Run 1.
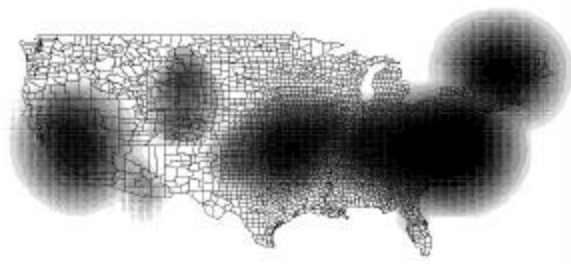


Figure 6.  GA Run 2.



Figure 7.  GA Run 3.

Figure 8.  GAM.

## 5.2 Comparison with other methods

GAM, the random circles method of Fotheringham and Zhan (1996), the case-point searching of Besag and Newell (1991), and the genetic algorithm of Conley *et al*. (2005) were run on the same dataset.

### 5.2.1 Did it find the same clusters?

Figures 9 through 11 show the results of the random, case-point, and genetic algorithm on the same dataset.  The implementations of these three methods are taken from the PROCLUDE software
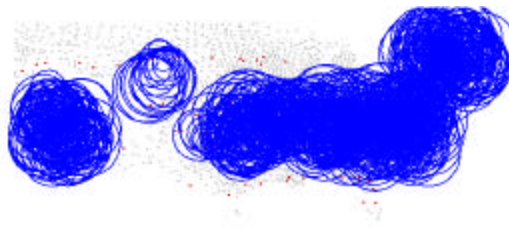


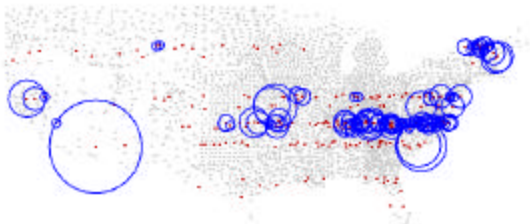Figure 9.  Conley *et al*.



Figure 10.  Fotheringham-Zhan.



Figure 11.  Besag-Newell.

As these figures demonstrate, the artificial life approach generally outperforms the genetic method from Conley *et al*. (2005) and Besag and Newell's method (1991).  The results from Fotheringham and Zhan's (1996) method are much closer to those of the geoboids.  However, given the banded nature of the dataset, perhaps the genetic method and Besag and Newell's method more closely reflect the true distribution of cases in the dataset, especially in separating the New England cases from the others in the east.

### 5.2.2 How long did it take?

The major obstacle with this method of determining optimal geoboid configurations is that the genetic algorithm used for parameter optimization took very long. As can be expected when the geoboids are run for 500 evaluations each for 20 configurations per generation for 20 generations, or 200,000 evaluations, each of which has its own internal genetic algorithm to optimize the non-spatial attributes of the cluster, the parameter optimization process requires large amounts of CPU time. The total running time for a single run of the genetic algorithm is approximately five hours on a machine with 2.00 GB of RAM and a 3.00 GHz Pentium 4 processor. This makes direct use of the genetic algorithm impractical in most situations.

## 6. Conclusions and Future Work

### 6.1 Usefulness of this system

While the lengthy running time limits this system's direct usefulness for individual applications, further research applying it to a diverse range of datasets may yield rules about how to configure the geoboids based directly on information about the dataset. For example, further research may establish that the ideal claustrophobia density could be directly calculated from the density of points in the dataset. Similarly, a sparse dataset could require higher maximum velocities. If this parameter optimization system can yield information on how to directly configure the geoboid system from the data rather than having to always go through the time-consuming parameter optimization step, then this system will prove to be useful.

### 6.2 Where do we go from here?

There are several future steps that need to be taken to complete this boid parameter optimization system. First, we need to implement the bitwise image comparison fitness measurement from section 4.2 so that we can evaluate how effective the ad hoc measurements are. Second, this system needs to be applied to a wide range of datasets. Only with this application to many varied datasets can the usefulness described above be reached. Also, there needs to be an analysis of how widely the results of different runs with the same configuration can vary. With this information, we can then determine how likely it is that the results from our single run of the configuration are representative of the configuration's overall performance. Finally, there needs to be a way to visualize how the parameters within the population of configurations change as the optimization algorithm runs. One way to do this is to use time series plots to show how the values of each of the parameters change and converge from one generation to the next. By combining this with similar plots of the statistics from each run, this visualization can also help to analyze how each of the parameters can affect the outcome of the boid run.

## 7. Acknowledgements

## 8. References

Agar, M., 2003,  My kingdom for a function: Modeling misadventures of the innumerate. *Journal of Artificial Societies and Societal Simulation*, **6**, online at http://jasss.soc.sur rey.ac.uk/6/3/8.html

Anselin, L., Cohen, J., Cook, D., Gorr, W., and Tita, G., 2000,  Spatial Analyses of Crime. In *Criminal Justice 2000 Volume 4: Measurement and Analysis of Crime and Justice* (Washington, DC: National Institute of Justice), pp. 213-262.

Besag, J. and Newell, J., 1991,  The detection of clusters in rare diseases. *Journal of the Royal Statistical Society, Series A*, **154**, 143-155.

Conley, J. F., Gahegan, M. N. and Macgill, J. (2005)  A genetic approach to detecting clusters in point datasets. *Geographical Analysis*, **37**, 286-314.

Cook, D. F., Ragsdale, C. T., and Major, R. L., 2000,  Combining a neural network with a genetic algorithm for process parameter optimization.  *Engineering Applications of Artificial Intelligence*, **13**, 391-396.

Deb, K., Anand, A., and Joshi, D., 2002,   A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, **10**, 371-395.

Dorling, D., 1998,   The epidemiology of the Liberal Democrat vote.  *Political Geography*, **17**, 45-70.

Duczmal, L. and Assunção, R., 2004,  A simulated annealing strategy for the detection of arbitrarily shaped spatial clusters. *Computational Statistics & Data Analysis*, **24**, 269-286.

Fotheringham, A. S. and Zhan, F. B., 1996,  A comparison of three exploratory methods for cluster detection in spatial point patterns. *Geographical Analysis*, **28**, 200-218.

Kulldorff, M., 1997,  A spatial scan statistic. *Communications in Statistics: Theory and Methods*, **26**, 1481-1496.

Macgill, J. and Openshaw, S., 1998,  The use of flocks to drive a Geographical Analysis Machine, In *International Conference on GeoComputation* (Bristol, UK).

Mantel, N., 1967, The detection of disease clustering and a generalized regression approach. *Cancer Research*, **27**, 209-220.

McKenna, J. E., Jr., 2003, An enhanced cluster analysis program with bootstrap significance testing for ecological community analysis. *Environmental Modelling & Software*, **18**, 205-220.

Openshaw, S., Charlton, M., Wymer, C. and Craft, A., 1987, A Mark 1 Geographical Analysis Machine for the automated analysis of point data sets. *International Journal of Geographic Information Systems*, **1**, 335-358.

Ragg, T., 2002, Bayesian learning and evolutionary parameter optimization. *AI Communications*, **15**, 61-74.

Reynolds, C. W., 1987, Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, **21**, 25-34.

Satpathy, H. P., 2003, Real-coded GA for parameter optimization in short-term load forecasting, In *IWANN 2003* (LCNS 2687), pp. 417-424.

Timander, L. M. and McLafferty., S., 1998, Breast cancer in West Islip, NY: A spatial clustering analysis with covariates. *Social Science and Medicine*, **46**, 1623-1635.

Vande Moere, A., 2004, Time-varying data visualization using information flocking boids, In *IEEE Symposium on Information Visualization 2004* (Austin, TX: IEEE), pp. 97-104.