To Kill a Flocking Bird

New Mexico
Supercomputing Challenge
Final Report
April 6, 2010

Team 70
Los Alamos High School

Team Members

Peter Ahrens
Stephanie Djidjev
Vicky Wang
Mei Liu

Teacher

Lee Goodwin

Project Mentors

Christine Ahrens
James Ahrens

Table of Contents

1.0 Executive Summary

This project explores which search techniques work best to optimize the parameters of a flocking model. Flocking is a natural phenomenon of many independent agents (birds) making decisions that lead to the group acting as a whole. The parameters used to control flocking are the angle at which a bird turns to get closer to his neighbors, the angle at which a bird turns to align itself with the rest of the flock and the angle at which a bird turns to get away from his neighbors if he is too close. NetLogo was used to develop an algorithm to judge qualities of a flock, implement the search techniques, run the search techniques and gather the data for comparison. The search techniques used were brute force (a test of all the possible combinations of parameters), genetic algorithms (a random search variant modeling natural selection), bracketing (dividing the search space iteratively), and steepest descent (searching locally and proceeding in the most promising direction to the solution from a random starting point in the search space). To evaluate a flock, a goodness function was created from the following functions: average distance to center, average difference in birds' distance to center, the average difference in the spacing of each bird to its nearest neighbor, and the average difference the birds' headings. A visual analysis of the brute force parameter study showed a diagonal gradient through the search space. The other search methods were tested, and compared based on the quality of the flocks produced, the reliability of the search, and the time efficiency. The results showed that the steepest descent technique had good performance and produced the best result.

2.0 Statement of the problem

Flocking is a natural phenomenon of many independent agents making decisions that lead to the group acting as a whole. Some examples of flocking behavior happen frequently in nature and they serve different purposes. Fish may exhibit flocking behavior to make themselves look bigger and ward off predators. Birds exhibit flocking behavior when they migrate. They rotate out of the front position in the flock and thus conserve energy breaking the wind. Elephants also exhibit flocking behavior for a different purpose. The larger elephants form a ring around the smaller, weaker elephants in an attempt to keep them safe from predators. Their flocks do not move much.

Flocking occurs in the manmade world as well. Flocking can be seen in strategic military formations and it can also be seen in traffic patterns (people tend to follow one another on large freeways). The principles of flocking can be applied to collision detection in robotic domains. Robots are usually programmed to avoid other robots or obstacles (unless they are battle bots). Flocking principles can also be applied to military applications with computer driven vehicles.

An interesting thing about flocking is that a computer can model it. Each decision-making entity, an "**agent**", begins in a random position, then using the location of its neighbors, makes decisions as to where to move itself. These decisions are usually called **cohesion**, **alignment** and **separation**. To cohere, an agent will move itself closer towards its neighbors. To align, an agent will align its heading with that of its neighbors. To separate, an agent will move itself away from its neighbors if it is too close. If these decisions are carefully balanced, the agents in the model will form a flock after a number of time steps. Balancing these decisions is a challenge and the flock quality is directly dependent upon the balance. Measuring the quality of the flock is a subjective process. Not everyone will agree that the quality of a flock is the same.

Usually the flocking decisions cohere, align, and separate are the parameters of the flocking model. How would you find the best parameters? The only certain way to do this would be to

test them all in all their combinations (thousands), and evaluate the resulting flocks using a function that tells you the quality of the flock or if it is a flock. This is a **brute-force** method of optimizing the parameters. This is computationally intensive and inefficient in its use of time, but it is the most accurate way of finding the best parameters.

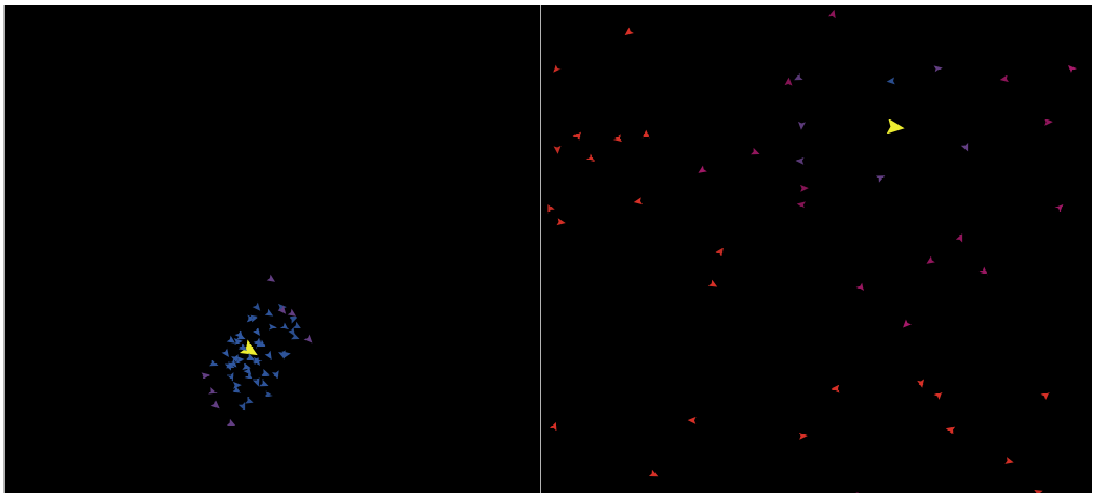
Many common search techniques used widely in computer science can be used for finding parameters, however not all of them are best suited towards flocking. Flocking can be unpredictable and time-intensive to search through and thus not all search techniques will perform the same. This project aims to find out which common search techniques will perform the best, be the most accurate and be most reliable in finding the optimal parameters to a flocking model. Others who are building or using flocking models can use this research.

3.0 Description of the Method Used to Solve the Problem

The method used to solve the problem of finding the best parameter search method started with a simple flocking model. This model was modified the model to evaluate the "goodness" of the flock by using **goodness functions** created by the team. After testing the goodness function visually, a brute-force method was used to understand the search space. Three search methods were then developed and tested, to find which produced the best parameter combination. The brute-force and other search methods were compared using statistics.

3.1 The NetLogo Flocking Model

A NetLogo model of flocking was found in the sample models included with NetLogo. NetLogo® was chosen because it is perfectly suited to flocking models. NetLogo® is an agent-based program and it is iterative. There is a graphical user interface and a display that shows the agents flocking and the time steps so far. The original code did not include any plots and all the parameters were controlled by sliders (the user). There were no flock quality evaluation functions included. In this model, the agents look like birds and flocked in a direction (they are moving as they flock), which is similar to the way that birds flock. The agents will be referred to as birds hereafter.



Picture 1: On the left, a flocking behavior is seen, on the right, there is no flocking.

The parameters used to control the decisions the birds made in this model **are max-cohere-turn**, **max-align-turn**, **max-separate-turn** and **minimum-separation**. Max-cohere-turn designates the maximum angle a bird can turn to cohere. It is the same with max-align-turn and max-separate-turn; they are also the maximum angle a bird can turn to make a flocking decision (align or separate). Minimum-separation is the distance between birds that signals that they need to turn away from each other. Originally, minimum separation was thought a parameter to be optimized, but was then understood to be a preference, as it designates how big a flock will be, which does not affect flock quality.

3.2 Evaluation and Goodness Functions

The first step was to develop a way to evaluate the flock. The goal was to have one goodness function that would tell how good the flock was. The goodness function used turned out to be an average of several evaluation functions. The evaluation functions developed were **mean distance to center**, **mean deviation of agents' distance to center**, the **mean deviation in the spacing of each agent to its nearest neighbor**, and the **mean deviation of the agents' headings**.

In order to evaluate the goodness of the flock, it was necessary to calculate the center of the flock. Although it was initially difficult to calculate the center of the birds in a boundless domain, a center of mass algorithm [7] was used. Using this algorithm, a **center bird** was created. It is enlarged for better visibility. Since it is the biggest bird of them all, it is colored yellow in honor of Big Bird. The center bird represents the center of the flock and its heading is the average heading of the flock.

The first evaluation function we developed was quite simple. It was the mean distance to center, which is exactly what the name implies. It is the average of all the distances from each bird to the center bird.

$$mean\ distance\ to\ center = \frac{\sum_{n=0}^{pf}\left(\sqrt{(x_n - x_c)^2 + (y_n - y_c)^2}\right)}{pf}$$

Equation 1: Where pf is the population of the flock, x_n is the x-coordinate of the n^{th} bird, x_c is the x-coordinate of the center, y_n is the y-coordinate of the n^{th} bird, and y_c is the y-coordinate of the center.

It indicates how clustered the birds are. A low value means the birds are clustered close to the center. A high value means they are spread out over the whole domain. A zero value indicates the birds are all in the same position (in a dot). This function was not used in the final code, because if the model is optimized completely to this function, the birds will be centered in a tiny dot.

The second evaluation function developed was the mean deviation of the birds' distance to center. This evaluation function measures the deviations in the distances between each bird and the center bird. This measurement will show how spread out and randomly spaced (within the flock) the birds are in relation to the center and each other.

$$mean\ deviation\ of\ distance\ to\ center = \frac{\sum_{m=0}^{pf}\left(abs(\sqrt{(x_m - x_c)^2 + (y_m - y_c)^2} - mean\ distance\ to\ center)\right)}{pf}$$

Equation 2: Where pf is the population of the flock, x_m is the x-coordinate of the m^{th} bird, x_c is the x-coordinate of the center, y_m is the y-coordinate of the m^{th} bird, and y_c is the y-coordinate of the center.

A low value indicates less deviation, which means the birds are more ordered. A high value indicates clumping (multiple clusters of birds) and a non-optimal flock. A zero value (theoretical max optimization) means the birds are either in a circle centered around the center bird or all at the center point, but since this is usually never achieved, it make this an overall useful evaluation function.

The third evaluation function developed was the mean difference in heading. This is what the name implies. It measures the average deviations in each birds' heading compared to the average flock heading.

$$mean\ heading\ deviation = \frac{\sum_{n=0}^{pf}(\arctan\left(\frac{\sum_{i=0}^{pf}\sin(h_i)}{\sum_{j=0}^{pf}\cos(h_j)}\right) - h_n)}{pf}$$

Equation 3: Where pf is the population of the flock, h_n is the heading of the n^{th} bird, h_i is the heading of the i^{th} bird, and h_j is the heading of the j^{th} bird.

While this function does not exactly measure how good a flock is spatially, it does show that the flock is not bumping into each other or going different directions. A low value indicates the birds are all heading in the same direction. A high value indicates the birds are running into each other in a central flock, not flocking, or going different directions. A zero value (theoretical max

optimization) means they are all going the same direction exactly, which does not necessarily indicate a good flock, but it does mean the birds are not running into each other.

The fourth evaluation function developed is the mean difference in spacing between each bird and its nearest neighbor.

$$mean\ spacing\ distance = \frac{\sum_{l=0}^{pf}\left(\sqrt{(x_l - x_{nn})^2 + (y_l - y_{nn})^2}\right)}{pf}$$

Equation 4: Where pf is the population of the flock, x_l is the x-coordinate of the l^{th} bird, x_{nn} is the x-coordinate of this bird's nearest neighbor, y_l is the y-coordinate of the l^{th} bird, and y_{nn} is the y-coordinate of the birds nearest neighbor.

$$\text{mean spacing deviation} = \frac{\sum_{q=0}^{pf}\left(abs(\sqrt{(x_q - x_{nn})^2 + (y_q - y_{nn})^2} - mean\ spacing\ distance)\right)}{pf}$$

Equation 5: Where pf is the population of the flock, x_q is the x-coordinate of the q^{th} bird, x_{nn} is the x-coordinate of this bird's nearest neighbor, y_q is the y-coordinate of the q^{th} bird, and y_{nn} is the y-coordinate of the birds nearest neighbor.

It measures the mean difference in the distances between these birds and the birds closest to it. This function is very effective at measuring even spacing. A low value indicates the birds have evenly spaced themselves in relation to each other. A high value indicates the birds are either clumping or not flocking. A zero value indicates the birds are in an isometric dot pattern or a dot in the center of the screen.

All of these evaluation functions have shortcomings if used exclusively, but if averaged, they produce an accurate measurement of the quality of a flock. The average of these evaluation functions is our goodness function. After time was spent studying the effectiveness of the functions visually, the functions were weighted at the values in Table 1.

Evaluation Functions	Weight
Mean distance to center	0.7
Mean deviation in the distances to center	0.75
Mean deviation in spacing between birds	1.0
Mean deviation of the agents' headings	1.0

Table 1. Weighting of evaluation functions within goodness function.

3.4 Brute Force Parameter Study

Before any search methods were run, a brute force search was done to understand the search space better. To do this, a NetLogo tool called "Behavior Space" was used. It is a tool designed to run parameter searches and similar tasks. It ran the flocking code for 200 iterations for each combination of the input parameters on a thirty-bird flock. For minimum-separation, the value was a constant 0.75. Note that minimum separation is just a preference for how big the end flock should be. For a 30-bird flock, .75 is a sufficient minimum separation, accounting for a 72-square-unit domain. For max- {cohere, align and separate}-turn they were increments of 1 between 0 and 10. Those were the original ranges offered by the interface to the original flocking code and anything outside that range produces a bad flock. Even though the parameters for the angles could be from 0 to 180, the goal is to optimize between 0 and 10. Each parameter combination was run once. The Behavior Space tool output a comma-separated file containing which combination of parameters was used and what the goodness function value was for that combination of parameters. This file was read into Microsoft Excel, edited to remove irrelevant data, saved as comma-separated file again, renamed to a ".particle" (ParaView compatible) file and read into ParaView [11] for analysis by visualization.

3.5 Other Search Method Implementations

Three parameter search methods were to be used: **bracketing**, **steepest descent** and **genetic algorithms**. These search methods were chosen because they are some of the most widely used and applicable to flocking.

A framework was used in all three optimizations to make the search methods comparable and for code reuse. The framework consisted of the general parameter search steps: **generate**, **evaluate** and **select**. The generate step takes in a tuple of parameter combination and parameter bounds (call it a "**state"**) and generates multiple states to be tested. These states are tested in the next step, evaluation. To evaluate, the flocking model is run for a set number of iterations and the result of the goodness function is coupled to each state. Finally, in the select step, the state with the best goodness function is selected and if the simulation is allowed to continue, it is fed back into the generate step, otherwise, this last state is the output. Each search method has its own stopping criteria.

The first parameter search technique implemented was bracketing [9]. This technique divides the search space in half and finds the best half, then uses the best half as the starting point for the next iteration, until the remaining half is small enough. In the generate step of bracketing, for each parameter, the min and the max were averaged to produce Point B. The average of the min and Point B, Point A, was calculated and the average of Point B and the max, Point C was calculated. A list of all the possible combinations of Point A and Point C for all four parameters is generated. To evaluate, the simulation is run for a specified number of iterations and the goodness function result is coupled with the parameter combination used. To select, the parameter combination with the lowest goodness function value is selected. For each parameter, if Point A was better, then the max is set as Point B. If Point C was better, the min is set at Point B. These min and max values are used for the next iteration. The stopping condition developed for this search technique was that after four iterations of gen-eval-select, it would stop. After four iterations, the values were precise enough for the parameter range used (0 to 10 for each parameter).

The second parameter search technique implemented was steepest descent [8, 9]. This technique starts at a random point in the search space and evaluates the local surrounding search space, then proceeds one step towards the most promising direction. To generate in the steepest descent parameter search method, all the possible combinations of each parameter value being incremented one step up or one step down are generated. Step size was set at .05 times the max value for that parameter. Each of these combinations is evaluated like in bracketing and each

combination is coupled with its goodness function value. To select, the best combination is sent back as input to the generate step. It stops when it generates a flock with a goodness function value under 0.1. This is the value deemed "good-enough."

The third search technique implemented is a genetic algorithm [1, 8]. It works by considering parameters to be genes that can be mutated, starting with organisms with randomly generated genes, evaluating them and choosing the organism with the best value to live, and others to die and be replaced by a mutation of the genes of the best organism. The generate step takes a pre-selected list of the worst in the flock, changes their parameters to be that of mean of the best birds and applies a mutation to that. The evaluate step runs like in the previous search techniques. To select, the best in the flock are selected and sent on to the generate stage. It stops when it generates a flock with a goodness function value under 0.1. This is the value deemed "good-enough."

4.0 Results

The purpose of the experiment was to find the optimal parameter search technique for flocking. Therefore, a brute force study was run to understand the search space. Subsequently, the various parameter search techniques were run, evaluated and compared.

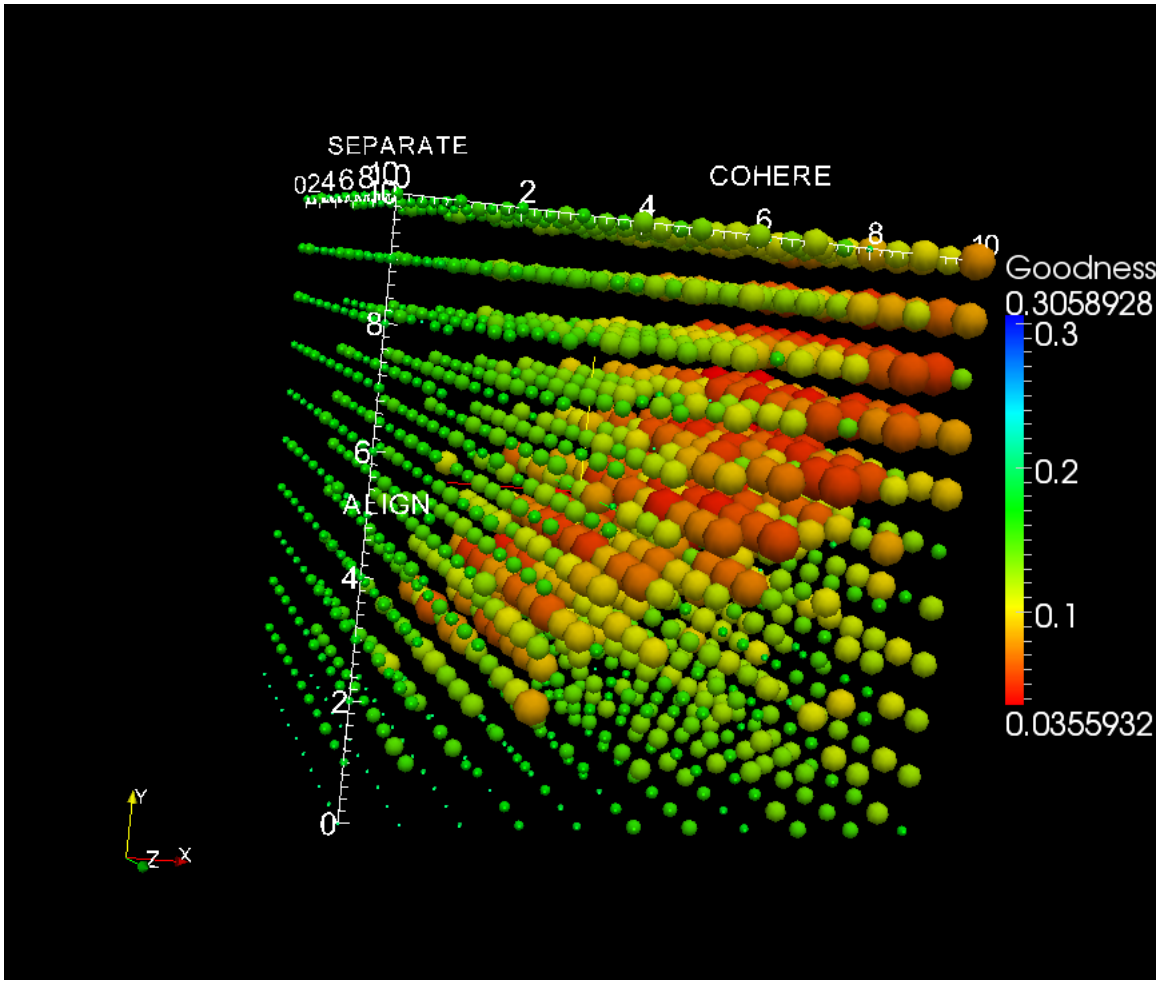


Figure 1. The parameter search space discovered via brute-force. Sphere radii and color represent goodness values.

In Figure 1, the results of the brute-force parameter study are shown. The radii of the spheres are inversely proportionate to the goodness value (the lower the goodness value, the better the flock, the bigger the spheres the better the flock). Different parameters are shown on each axis (cohere, align and separate). The colors also signify goodness, with red being the best flock, blue being the worst. The figure shows that when cohere and align are approximately equal in value, a better goodness value is found. From this figure, a parameter search space with a diagonal gradient can be seen.

.

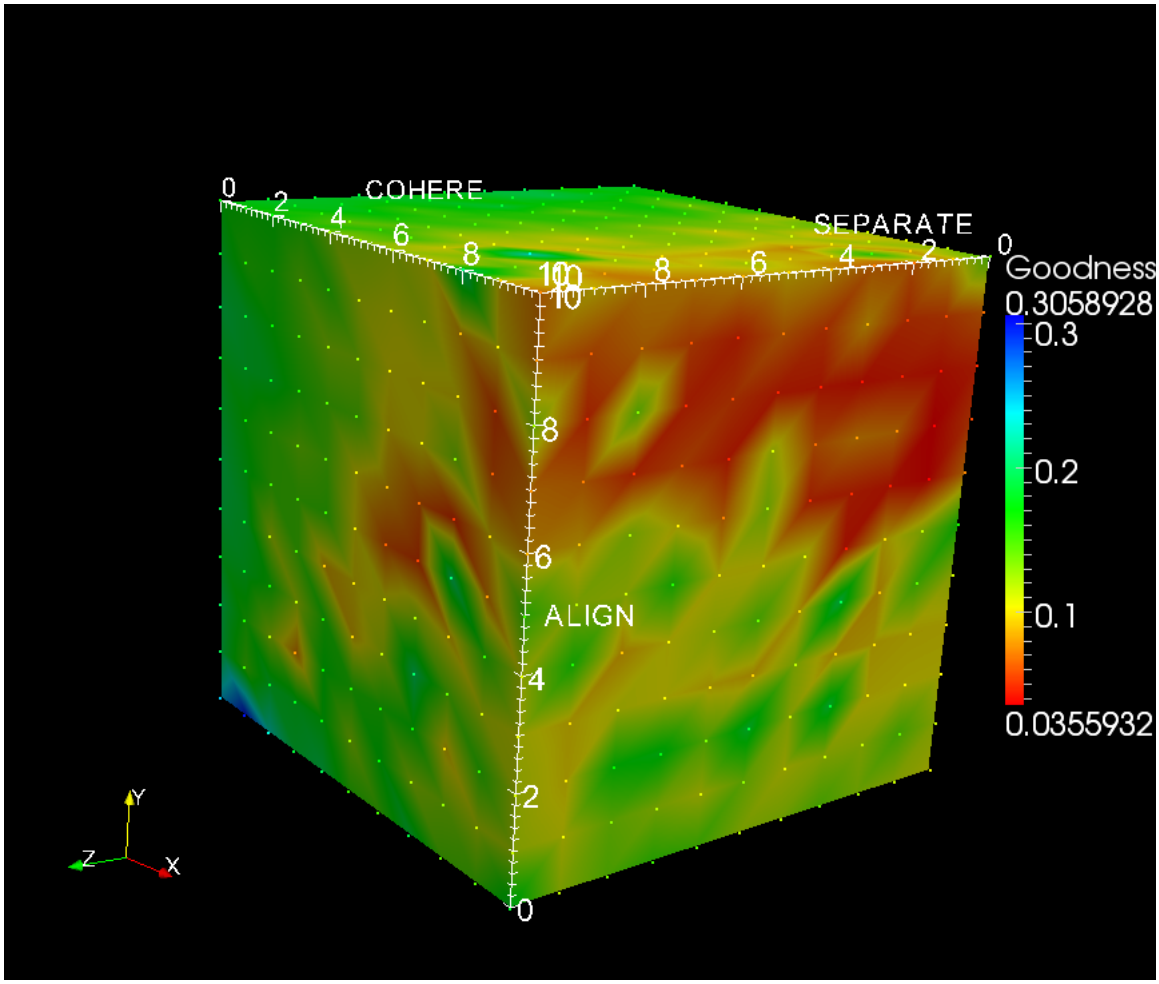


Figure 2. Search space as interpolated surface cube.

Using the same test harness, Behavior Space, each parameter search technique was run ten times and the outputs saved to a file. The outputs included the time it took to run each search, the parameters to the flocking run, the average goodness function value, and the number of "gen-eval-select" steps. Using this data, and the code written, the number of evaluations (200-timestep flocking runs) is calculated.

Results	Brute Force	Bracketing	Genetic	Steepest Descent
Time (Min)	~150	8.79850	2.06589	2.93913
Reliability	100%	70%	90%	100%
Goodness	0.03559	0.05466	0.08721	0.06869
Evaluations	1331	32	8.0	28.8

Table 2. Comparison of parameter search techniques.

- Time – The average time in minutes it takes for the search to come to a verdict.
- Reliability – The percentage of successful runs (runs having a goodness function under 0.1 in at least 30 iterations of the search).
- Goodness – The output flock goodness of each successful run of the search technique. For the bracketing, genetic and steepest descent, this is an average of the 10 runs output.
- Evaluations – The average number of 200 iteration flocking tests that are run (in successful runs).

5.0 Conclusions

The comparison of parameter search techniques shows that steepest descent is the most overall useful search technique, but each has its strengths and weaknesses.

Steepest descent is most "reliable" (as defined above) most likely because the parameter search space appears to be devoid of local minima and has a broad gradient for steepest descent to follow. It also has very good performance.

Bracketing was the least reliable, but the average goodness value of its output parameter combination is the closest to brute-force output. It performance is worse than steepest descent and genetic, but still much better than brute-force. Bracketing's reliability appeared to be impacted negatively by inadequate sampling of the search space due to the position of the gradient.

Although the genetic search technique is intriguing and its performance was better than the other three techniques, it average goodness value of its output parameter combination was the worst of the four. Since this technique was the only one that incorporated randomness, that may have affect its output goodness negatively.

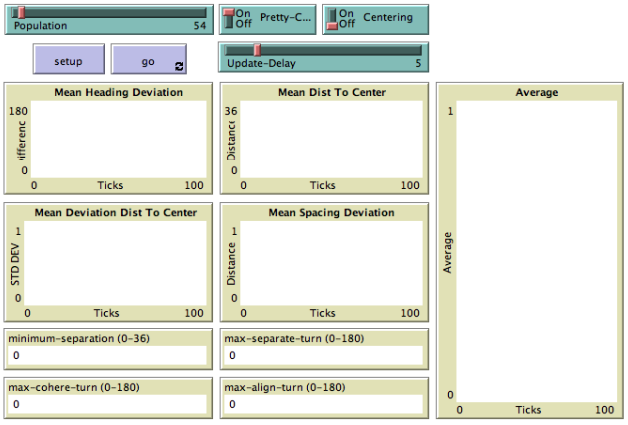
6.0 Significant Original Achievement

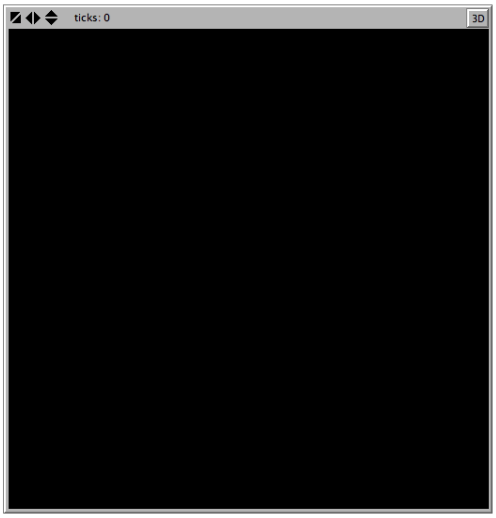
The most significant original achievement that was made by Team 70 was understanding the parameter search space. This is important to all of those interested in flocking, as the search space is very important to the search techniques run on it. Furthermore in order to understand the search space, the team made original contributions in equations to evaluate the flock.

7.0 Work Products

The code for the Behavior Space was stored with the NetLogo model. Each search technique has a different code base, but the goodness functions are identical. Code from original model is marked. BehaviorSpace code could not be included as it is stored in a GUI.

7.1 Flocking with Goodness Functions





breed [centers center]

globals [
  vision
  mean-minimum-separation
  mean-max-separate-turn
  mean-max-cohere-turn
  mean-max-align-turn

  flock-center-x
  flock-center-y

```
    avg-spacing
    avg-flock-heading
]

turtles-own [
  minimum-separation
  max-separate-turn
  max-cohere-turn
  max-align-turn
  flockmates   ;; agentset of nearby turtles
  nearest-flockmate   ;; closest one of our flockmates
  nearest-neighbor   ;; closest turtle
  other-turtles   ;; turtles that are not the turtle in question

  dist-center   ;; the distance between the turtle and the center of the flock
  dev-heading   ;; difference between heading of turtle and the average heading
  dev-spacing   ;; difference between spacing from turtle to nearest neighbor and the average diff
spacing
  dev-dist-center   ;; difference between distance from turtle to center and the average distance
from turtle to center

  dist-nearest-neighbor
]

to setup ; this function modified from original code by team 70
  clear-all
  crt Population
    [ set color yellow - 2 + random 7  ;; random shades look nice
      set size 1.5  ;; easier to see
      setxy random-xcor random-ycor ]
  crt 1
    [ set breed centers
      set color blue
      set size 3.0
      setxy random-xcor random-ycor]
  ask turtles
    [ set minimum-separation 1.0
      set max-separate-turn 1.5
      set max-cohere-turn 5.0
      set max-align-turn 3.0]
  set vision 36.0
end

to go
  ask turtles [
```

```
  update-flocking-vars
  flock]
  ;; the following line is used to make the turtles
  ;; animate more smoothly.
  repeat 5 [ ask turtles [ fd 0.2 ] display ]
  ;; for greater efficiency, at the expense of smooth
  ;; animation, substitute the following line instead:
  ;;   ask turtles [ fd 1 ]
  ask centers [do-center-stuff]
  ifelse centering
    [ ask centers
        [ ride-me]]
    [ reset-perspective]
  tick
  if ticks mod update-delay = 0
  [ask turtles [
  update-colors
  update-plots
  update-goodness-fxn-vars
  update-parameters]]
end

;;; UPDATES

to update-plots
 set-current-plot "Mean Heading Deviation"
  set-current-plot-pen "heading"
  plotxy ticks mean-dev-heading
 set-current-plot "Mean Deviation Dist To Center"
  set-current-plot-pen "STD DEV"
  plotxy ticks mean-dev-dist-center
 set-current-plot "Mean Dist To Center"
  set-current-plot-pen "distance"
  plotxy ticks mean-dist-center
 set-current-plot "Mean Spacing Deviation"
  set-current-plot-pen "difference"
  plotxy ticks mean-dev-spacing
 set-current-plot "Average"
  set-current-plot-pen "avg-fxn"
  plotxy ticks average-flock-goodness
end

to update-colors
  if Pretty-Colors
      [set color approximate-rgb ((dist-center / 36) * 255) 0 (255 -((dist-center / 36) * 255))
      ask centers
```

```
    [set color yellow]]
end

to update-flocking-vars
  find-other-turtles
  find-flockmates
  find-nearest-neighbor
  find-flock-center-x
  find-flock-center-y
  find-avg-flock-heading
end

to update-goodness-fxn-vars
  find-dev-dist-center
  find-dev-heading
  find-dev-spacing
  find-dist-center
end

to update-parameters
  set mean-minimum-separation mean [minimum-separation] of turtles
  set mean-max-separate-turn mean [max-separate-turn] of turtles
  set mean-max-cohere-turn mean [max-cohere-turn] of turtles
  set mean-max-align-turn mean [max-align-turn] of turtles
end

;;; TURTLE QUALITY FUNCTIONS

to find-dev-dist-center
  set dev-dist-center abs (mean-dist-center - dist-center)
end

to find-dev-heading
  set dev-heading abs (avg-flock-heading - heading)
end

to find-dev-spacing
  set dist-nearest-neighbor distance nearest-neighbor
  set dev-spacing abs((mean[dist-nearest-neighbor] of turtles) - dist-nearest-neighbor)
end

to find-dist-center
  set dist-center distancexy flock-center-x flock-center-y
end

to-report average-turtle-goodness
```

```
  report (((1 * dev-heading / 360) + (1 * dev-spacing / 36) + (0.75 *  dev-dist-center / 36) + (0.7 *
dist-center / 36)) / 4)
end

;;; FLOCK QUALITY FUNCTIONS

to-report average-flock-goodness
  report (((1 * mean-dev-heading / 360) + (1 * mean-dev-spacing / 36) + (0.75 * mean-dev-dist-
center / 36) + (0.7 * mean-dist-center / 36))/ 4)
end

to-report mean-dist-center
  report mean [dist-center] of turtles
end

to-report mean-dev-heading
  report mean [dev-heading] of turtles
end

to-report mean-dev-dist-center
  report mean [dev-dist-center] of turtles
end

to-report mean-dev-spacing
  report mean [dev-spacing] of turtles
end

;;; FLOCK CHARACTERISTICS

to find-flock-center-x
  let r-i (max-pxcor / ( 2 * pi ))
  let x-i map [r-i * cos(((? * 2 * pi) / max-pxcor) * (180 / pi))] [xcor] of turtles
  let z-i map [r-i * sin(((? * 2 * pi) / max-pxcor) * (180 / pi))] [xcor] of turtles
  let tube-avg-i-x mean x-i
  let tube-avg-i-z mean z-i
  set flock-center-x ((atan (tube-avg-i-z) (tube-avg-i-x)) * (pi / 180)) * (max-pxcor / (2 * pi))
end

to find-flock-center-y
  let r-j (max-pycor / ( 2 * pi ))
  let y-j map [r-j * cos(((? * 2 * pi) / max-pycor) * (180 / pi))] [ycor] of turtles
  let z-j map [r-j * sin(((? * 2 * pi) / max-pycor) * (180 / pi))] [ycor] of turtles
  let tube-avg-j-y mean y-j
  let tube-avg-j-z mean z-j
  set flock-center-y ((atan (tube-avg-j-z) (tube-avg-j-y)) * (pi / 180)) * (max-pycor / (2 * pi))
end
```

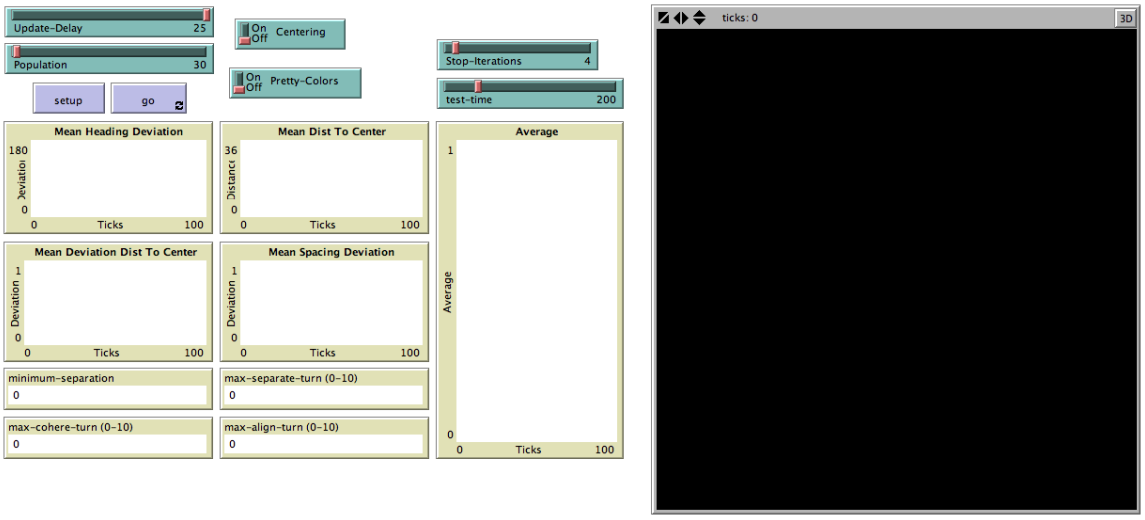
```
to find-avg-flock-heading
  set avg-flock-heading atan sum [sin heading] of turtles
                           sum [cos heading] of turtles
end

;; code from here on was not written by group 70

;;; FLOCKING

to flock  ;; turtle procedure
  if any? flockmates
    [ find-nearest-flockmate
      ifelse distance nearest-flockmate < minimum-separation
        [ separate ]
        [ align
          cohere ] ]
end

to find-flockmates  ;; turtle procedure
  set flockmates other-turtles in-radius vision
end

to find-other-turtles
  set other-turtles other turtles
end

to find-nearest-neighbor
  set nearest-neighbor min-one-of other-turtles [distance myself]
end

to find-nearest-flockmate ;; turtle procedure
  set nearest-flockmate nearest-neighbor
end

;;; SEPARATE

to separate  ;; turtle procedure
  turn-away ([heading] of nearest-flockmate) max-separate-turn
end

;;; ALIGN

to align  ;; turtle procedure
  turn-towards average-flockmate-heading max-align-turn
end
```

```
to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  ;; Theoretically this could fail if both sums are 0
  ;; since atan 0 0 is undefined, but in practice that's
  ;; vanishingly unlikely.
  report atan sum [sin heading] of flockmates
           sum [cos heading] of flockmates
end

;;; COHERE

to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates max-cohere-turn
end

to-report average-heading-towards-flockmates  ;; turtle procedure
  ;; "towards myself" gives us the heading from the other turtle
  ;; to me, but we want the heading from me to the other turtle,
  ;; so we add 180
  report atan mean [sin (towards myself + 180)] of flockmates
           mean [cos (towards myself + 180)] of flockmates
end

;;; CENTER BOID

to do-center-stuff
  setxy flock-center-x flock-center-y
  set heading avg-flock-heading
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings new-heading heading) max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading) max-turn
end

;; turn right by "turn" degrees (or left if "turn" is negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
```

```
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

; *** NetLogo 4.0.4 Model Copyright Notice ***
;
; This model was created as part of the project: CONNECTED MATHEMATICS:
; MAKING SENSE OF COMPLEX PHENOMENA THROUGH BUILDING OBJECT-BASED
PARALLEL
; MODELS (OBPML).  The project gratefully acknowledges the support of the
; National Science Foundation (Applications of Advanced Technologies
; Program) -- grant numbers RED #9552950 and REC #9632612.
;
; Copyright 1998 by Uri Wilensky.  All rights reserved.
;
; Permission to use, modify or redistribute this model is hereby granted,
; provided that both of the following requirements are followed:
; a) this copyright notice is included.
; b) this model will not be redistributed for profit without permission
;    from Uri Wilensky.
; Contact Uri Wilensky for appropriate licenses for redistribution for
; profit.
;
; This model was converted to NetLogo as part of the projects:
; PARTICIPATORY SIMULATIONS: NETWORK-BASED DESIGN FOR SYSTEMS
LEARNING
; IN CLASSROOMS and/or INTEGRATED SIMULATION AND MODELING
ENVIRONMENT.
; The project gratefully acknowledges the support of the
; National Science Foundation (REPP & ROLE programs) --
; grant numbers REC #9814682 and REC-0126227.
; Converted from StarLogoT to NetLogo, 2002.
;
; To refer to this model in academic publications, please use:
; Wilensky, U. (1998).  NetLogo Flocking model.
; http://ccl.northwestern.edu/netlogo/models/Flocking.
; Center for Connected Learning and Computer-Based Modeling,
; Northwestern University, Evanston, IL.
;
; In other publications, please use:
; Copyright 1998 Uri Wilensky.  All rights reserved.
; See http://ccl.northwestern.edu/netlogo/models/Flocking
; for terms of use.
```

;
; *** End of NetLogo 4.0.4 Model Copyright Notice ***

7.2 Bracketing




```
extensions[array]
breed [centers center]

globals [
  vision
  mean-minimum-separation
  mean-max-separate-turn
  mean-max-cohere-turn
  mean-max-align-turn

  flock-center-x
  flock-center-y

  avg-spacing
  avg-flock-heading

  goodness-output

  state
  gen-a
  gen-c
  state-min
  state-max
  state-gen
  state-eval
  current-param
```

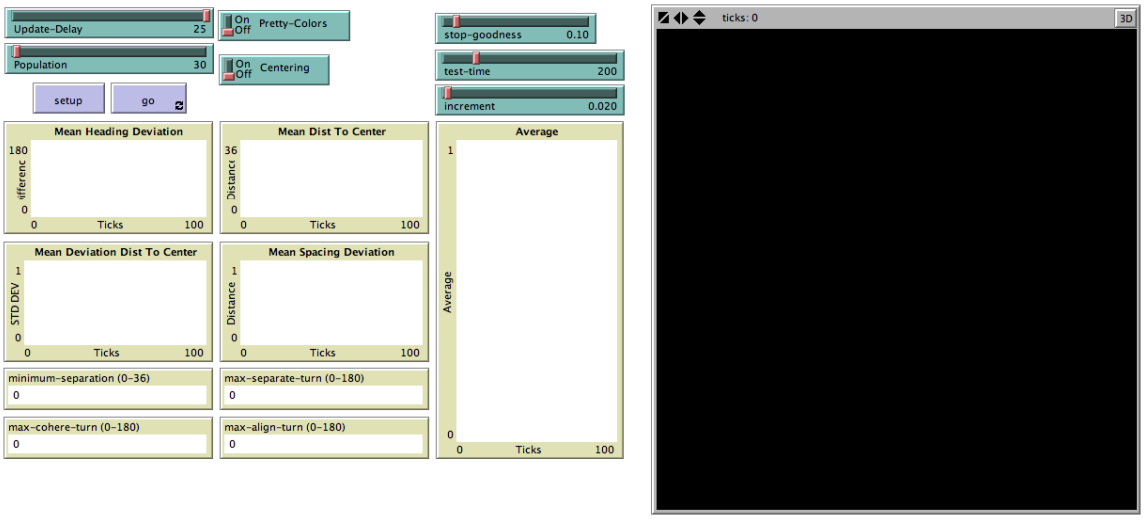
```
    bracket-b
    stopping
    iterations
]

turtles-own [
  minimum-separation
  max-separate-turn
  max-cohere-turn
  max-align-turn
  flockmates   ;; agentset of nearby turtles
  nearest-flockmate   ;; closest one of our flockmates
  nearest-neighbor   ;; closest turtle
  other-turtles   ;; turtles that are not the turtle in question

  dist-center   ;; the distance between the turtle and the center of the flock
  dev-heading   ;; difference between heading of turtle and the average heading
  dev-spacing   ;; difference between spacing from turtle to nearest neighbor and the average diff
spacing
  dev-dist-center   ;; difference between distance from turtle to center and the average distance
from turtle to center

  dist-nearest-neighbor
]

to setup ; this function modified from original code by team 70
  clear-all
  crt Population
    [ set color yellow - 2 + random 7  ;; random shades look nice
      set size 1.5  ;; easier to see
      setxy random-xcor random-ycor ]
  crt 1
    [ set breed centers
      set color blue
      set size 3.0
      setxy random-xcor random-ycor]
  ask turtles
    [ set minimum-separation 0.75]
  set vision 36.0
  set test-time 200
  set iterations 0
  set stopping False
  set gen-a array:from-list [0 0 0]
  set gen-c array:from-list [0 0 0]
  set state-min array:from-list [0 0 0]
  set state-max array:from-list [10 10 10]
```

```
  set state array:from-list (list (random array:item state-max 0) (random array:item state-max 1)
(random array:item state-max 2))
end

to go
  gen
  eval
  select
  are-we-done?
end




to gen
  set state-gen []
  set current-param 0
  foreach array:to-list state
    [ set bracket-b ((item current-param array:to-list state-min) + (item current-param array:to-list
state-max)) / 2
      let state1 array:to-list state
      let state2 array:to-list state
      set state1 replace-item current-param state1 ((bracket-b + (array:item state-min current-
param)) / 2)
      set state2 replace-item current-param state2 ((bracket-b + (array:item state-max current-
param)) / 2)
      array:set gen-a current-param array:item array:from-list state1 current-param
      array:set gen-c current-param array:item array:from-list state2 current-param
      set current-param current-param + 1]
  let possible-state [0 0 0]
  foreach list (array:item gen-a 0) (array:item gen-c 0)
    [ set possible-state replace-item 0 possible-state ?
      foreach list (array:item gen-a 1) (array:item gen-c 1)
        [ set possible-state replace-item 1 possible-state ?
          foreach list (array:item gen-a 2) (array:item gen-c 2)
            [ set possible-state replace-item 2 possible-state ?
              set state-gen lput array:from-list possible-state state-gen]]]
  set state-gen array:from-list state-gen

end



to eval
  set state-eval []
```

```
  let eval-run 0
  foreach array:to-list state-gen
    [ ask turtles [update-flocking-vars]
      set state (array:item state-gen eval-run)
      ask turtles [set max-separate-turn array:item state 0]
      ask turtles [set max-cohere-turn array:item state 1]
      ask turtles [set max-align-turn array:item state 2]
      reset-ticks
      clear-all-plots
      ask turtles [setxy random-xcor random-ycor]
      while [ticks < test-time]
        [ ask turtles [
          update-flocking-vars
          flock]
          ;; the following line is used to make the turtles
          ;; animate more smoothly.
          repeat 5 [ ask turtles [ fd 0.2 ] display ]
          ;; for greater efficiency, at the expense of smooth
          ;; animation, substitute the following line instead:
          ;;   ask turtles [ fd 1 ]
          ask centers [do-center-stuff]
          ifelse centering
            [ ask centers
               [ ride-me]]
            [ reset-perspective]
          tick
          if ticks mod update-delay = 0
          [ask turtles [
          update-colors
          update-plots
          update-goodness-fxn-vars
          update-parameters]]]
      ask turtles [
          update-colors
          update-plots
          update-goodness-fxn-vars
          update-parameters]
      set state-eval lput average-flock-goodness state-eval
      set eval-run eval-run + 1]
end

to select
  let i 0
  let best-state 0
  foreach state-eval
    [ if min state-eval = ?
```

```
      [ set best-state array:item state-gen i
          set goodness-output ?]
      set i i + 1]
  set current-param 0
  foreach array:to-list best-state
    [ set bracket-b ((item current-param array:to-list state-min) + (item current-param array:to-list
state-max)) / 2
      ifelse ? = array:item gen-a current-param
        [ array:set state-max current-param bracket-b]
        [ array:set state-min current-param bracket-b]
      set current-param current-param + 1 ]

  set iterations iterations + 1
end

to are-we-done?
  if iterations >= stop-iterations
    [ set goodness-output average-flock-goodness
      set stopping True]
end

;;; UPDATES
;; original code previously listed above omitted here for this document
```

7.3 Steepest Descent




```
extensions[array]
breed [centers center]

globals [
   vision
   mean-minimum-separation
```

```
    mean-max-separate-turn
    mean-max-cohere-turn
    mean-max-align-turn

    flock-center-x
    flock-center-y

    avg-flock-heading

    state
    gen-a
    gen-c
    state-min
    state-max
    state-gen
    state-eval
    best-state
    current-param

    stopping
    goodness-output
]

turtles-own [
  minimum-separation
  max-separate-turn
  max-cohere-turn
  max-align-turn
  flockmates   ;; agentset of nearby turtles
  nearest-flockmate   ;; closest one of our flockmates
  nearest-neighbor   ;; closest turtle
  other-turtles   ;; turtles that are not the turtle in question

  dist-center   ;; the distance between the turtle and the center of the flock
  dev-heading   ;; difference between heading of turtle and the average heading
  dev-spacing   ;; difference between spacing from turtle to nearest neighbor and the average diff
spacing
  dev-dist-center   ;; difference between distance from turtle to center and the average distance
from turtle to center

  dist-nearest-neighbor

]

to setup ; this function modified from original code by team 70
  clear-all
```

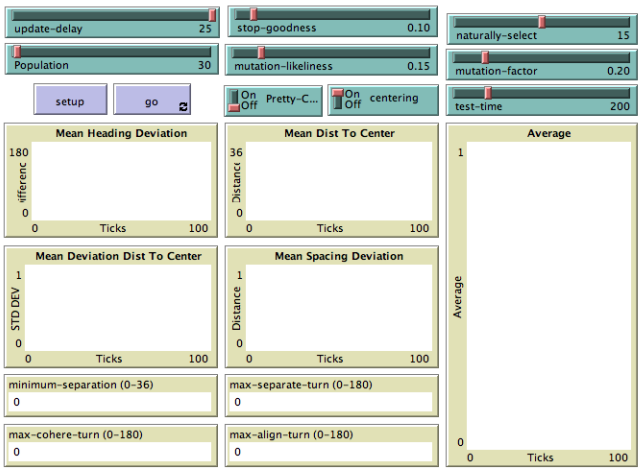
```
  crt Population
    [ set color yellow - 2 + random 7  ;; random shades look nice
      set size 1.5  ;; easier to see
      setxy random-xcor random-ycor ]
  crt 1
    [ set breed centers
      set color blue
      set size 3.0
      setxy random-xcor random-ycor]
  ask turtles
    [ set minimum-separation 1.0
      set max-separate-turn 1.5
      set max-cohere-turn 5.0
      set max-align-turn 3.0]
  set vision 36.0
  ask turtles [set minimum-separation 0.75]
  set state-min array:from-list [0 0 0]
  set gen-a array:from-list [0 0 0]
  set gen-c array:from-list [0 0 0]
  set state-max array:from-list [10 10 10]
  set state array:from-list (list (random array:item state-max 0) (random array:item state-max 1)
(random array:item state-max 2))
  set stopping False
  set goodness-output 0
end

to go
  gen
  eval
  select
  are-we-done?
end




to gen
  set state-gen []
  set current-param 0
  foreach array:to-list state
    [ let state1 array:to-list state
      let state2 array:to-list state
      set state1 replace-item current-param state1 (array:item state current-param + (increment *
array:item state-max current-param))
      if item current-param state1 < 0
```

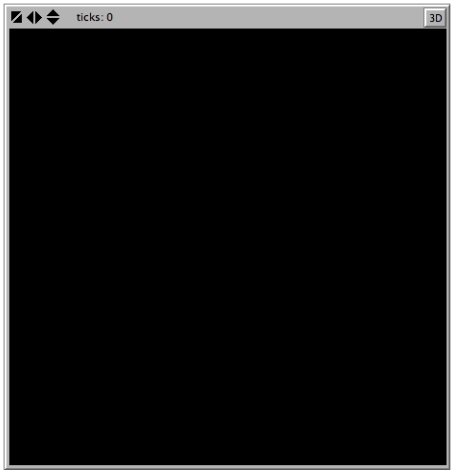
```
      [ set state1 replace-item current-param state1 0]
    if item current-param state1 > array:item state-max current-param
      [ set state1 replace-item current-param state1 array:item state-max current-param]
    set state2 replace-item current-param state2 (array:item state current-param - (increment *
array:item state-max current-param))
    if item current-param state2 < 0
      [ set state2 replace-item current-param state2 0]
    if item current-param state2 > array:item state-max current-param
      [ set state2 replace-item current-param state2 array:item state-max current-param]
    array:set gen-a current-param array:item array:from-list state1 current-param
    array:set gen-c current-param array:item array:from-list state2 current-param
    set current-param current-param + 1]
  let possible-state [0 0 0]
  foreach list (array:item gen-a 0) (array:item gen-c 0)
   [ set possible-state replace-item 0 possible-state ?
    foreach list (array:item gen-a 1) (array:item gen-c 1)
     [ set possible-state replace-item 1 possible-state ?
       foreach list (array:item gen-a 2) (array:item gen-c 2)
         [ set possible-state replace-item 2 possible-state ?
           set state-gen lput array:from-list possible-state state-gen]]]
  set state-gen array:from-list state-gen

end



to eval
  set state-eval []
  let eval-run 0
  foreach array:to-list state-gen
   [ ask turtles [update-flocking-vars]
    set state (array:item state-gen eval-run)
    ask turtles [set max-separate-turn array:item state 0]
    ask turtles [set max-cohere-turn array:item state 1]
    ask turtles [set max-align-turn array:item state 2]
    reset-ticks
    clear-all-plots
    ask turtles [setxy random-xcor random-ycor]
    while [ticks < test-time]
     [ ask turtles [
       update-flocking-vars
       flock]
       ;; the following line is used to make the turtles
       ;; animate more smoothly.
       repeat 5 [ ask turtles [ fd 0.2 ] display ]
       ;; for greater efficiency, at the expense of smooth
```

```
        ;; animation, substitute the following line instead:
        ;;   ask turtles [ fd 1 ]
        ask centers [do-center-stuff]
        ifelse centering
          [ ask centers
            [ ride-me]]
          [ reset-perspective]
        tick
        if ticks mod update-delay = 0
        [ask turtles [
        update-colors
        update-plots
        update-goodness-fxn-vars
        update-parameters]]]
      ask turtles [
        update-colors
        update-plots
        update-goodness-fxn-vars
        update-parameters]
      set state-eval lput average-flock-goodness state-eval
      set eval-run eval-run + 1]
end

to select
  let i 0
  set best-state 0
  foreach state-eval
    [ if min state-eval = ?
        [ set best-state array:item state-gen i
          set goodness-output ?]
      set i i + 1]
  set state best-state
end

to are-we-done?
  if min state-eval < stop-goodness
    [ ask turtles [set max-separate-turn array:item best-state 0]
      ask turtles [set max-cohere-turn array:item best-state 1]
      ask turtles [set max-align-turn array:item best-state 2]
      set goodness-output min state-eval
      set stopping True]
end
```

;;; UPDATES
;; original code previously listed above omitted here for this document

7.4 Genetic





extensions[array]
breed [centers center]

globals [
  vision
  mean-minimum-separation
  mean-max-separate-turn
  mean-max-cohere-turn
  mean-max-align-turn

  flock-center-x
  flock-center-y

  avg-spacing
  avg-flock-heading

  outer-birds
  unlucky-turtles
  lucky-turtles

  state
  state1
  state2
  state-min
  state-max
  state-gen
  state-eval
  current-param

```
    goodness-output
    stopping
]
turtles-own [
  minimum-separation
  max-separate-turn
  max-cohere-turn
  max-align-turn
  flockmates   ;; agentset of nearby turtles
  nearest-flockmate   ;; closest one of our flockmates
  nearest-neighbor   ;; closest turtle
  other-turtles   ;; turtles that are not the turtle in question
  parents

  dist-center   ;; the distance between the turtle and the center of the flock
  dev-heading   ;; difference between heading of turtle and the average heading
  dev-spacing   ;; difference between spacing from turtle to nearest neighbor and the average diff
spacing
  dev-dist-center   ;; difference between distance from turtle to center and the average distance
from turtle to center
  dev-heading-history

  dist-nearest-neighbor
]

to setup ; this function modified from original code by team 70
  clear-all
  crt Population
    [ set color yellow - 2 + random 7  ;; random shades look nice
      set size 1.5  ;; easier to see
      setxy random-xcor random-ycor ]
  crt 1
    [ set breed centers
      set color blue
      set size 3.0
      setxy random-xcor random-ycor]
  ask turtles [set minimum-separation 0.75]
  set vision 36.0
  set current-param 0
  set test-time 200
  set unlucky-turtles no-turtles
  set state-min array:from-list [0 0 0]
  set state-max array:from-list [10 10 10]
  set state array:from-list (list (random array:item state-max 0) (random array:item state-max 1)
(random array:item state-max 2))
  ask turtles [set max-separate-turn (random array:item state-max 0)]
```

```
  ask turtles [set max-cohere-turn (random array:item state-max 1)]
  ask turtles [set max-align-turn (random array:item state-max 2)]
  set stopping false

end

to go
  gen
  eval
  select
  are-we-done?
end

to gen
if any? unlucky-turtles
  [ ask unlucky-turtles
     [ set parents (turtle-set one-of lucky-turtles one-of lucky-turtles)
       set color yellow - 2 + random 7  ;; random shades look nice
       set size 1.5  ;; easier to see
       let parent-num1 random-float 1
       ifelse parent-num1 < mutation-likeliness
         [ set max-separate-turn (mean-max-separate-turn + (((random -2) * 2 + 1) * ((random-float
1) ^ (1 / mutation-factor)) * array:item state-max 0))]
         [ set max-separate-turn ([max-separate-turn] of one-of parents)]
       if max-separate-turn < 0
         [ set max-separate-turn 0]
       if max-separate-turn > array:item state-max 0
         [ set max-separate-turn array:item state-max 0]
       let parent-num2 random-float 1
       ifelse parent-num2 < mutation-likeliness
         [ set max-cohere-turn (mean-max-cohere-turn + (((random -2) * 2 + 1) * ((random-float 1)
^ (1 / mutation-factor)) * array:item state-max 1))]
         [ set max-cohere-turn ([max-cohere-turn] of one-of parents)]
       if max-cohere-turn < 0
         [ set max-cohere-turn  0]
       if max-cohere-turn > array:item state-max 1
         [ set max-cohere-turn array:item state-max 1]
       let parent-num3 random-float 1
       ifelse parent-num3 < mutation-likeliness
         [ set max-align-turn (max-align-turn + (((random -2) * 2 + 1) * ((random-float 1) ^ (1 /
mutation-factor)) * array:item state-max 2))]
         [ set max-align-turn ([max-align-turn] of one-of parents)]
       if max-align-turn < 0
         [ set max-align-turn 0]
       if max-align-turn > array:item state-max 2
         [ set max-align-turn array:item state-max 2]]]
```

```
end

to eval
  ask turtles [update-flocking-vars]
  reset-ticks
  clear-all-plots
  ;ask turtles [set dev-heading-history []]
  ask turtles [setxy random-xcor random-ycor]
  while [ticks < test-time]
    [ ask turtles [
      update-flocking-vars
      flock]
      ;; the following line is used to make the turtles
      ;; animate more smoothly.
      repeat 5 [ ask turtles [ fd 0.2 ] display ]
      ;; for greater efficiency, at the expense of smooth
      ;; animation, substitute the following line instead:
      ;;   ask turtles [ fd 1 ]
      ask centers [do-center-stuff]
      ifelse centering
        [ ask centers
            [ride-me]]
        [ reset-perspective]
      tick
      if ticks mod update-delay = 0
        [ ask turtles [
          update-colors
          update-plots
          update-goodness-fxn-vars
          update-parameters]]]
  ask turtles [
      update-colors
      update-plots
      update-goodness-fxn-vars
      update-parameters]
end

to select
  ask centers[set outer-birds other turtles]
  set unlucky-turtles max-n-of naturally-select outer-birds [average-turtle-goodness]
  set lucky-turtles min-n-of (Population - naturally-select) outer-birds [average-turtle-goodness]
end

to are-we-done?
  if average-flock-goodness < stop-goodness
    [ set mean-max-separate-turn mean [max-separate-turn] of lucky-turtles
```

```
    set mean-max-cohere-turn mean [max-cohere-turn] of lucky-turtles
    set mean-max-align-turn mean [max-align-turn] of lucky-turtles
    set goodness-output average-flock-goodness
    set stopping True]
  set goodness-output average-flock-goodness
end

;;; UPDATES
;; original code previously listed above omitted here for this document
```

8.0 Bibliography

[1] Stephanie Forrest: Genetic Algorithms: Principles of Natural Selection Applied to Computation, 2007

[2] Isaac Councill, Lee Giles : Random Search For Multiple Layer Perceptron 2005

[3] István Maros : Computational Techniques of the *Simplex Method* (International Series in Operations Research & Management Science) 2004

[4] R. Fletcher: Practical Methods of Optimization 2005

[5] Nigel Gilbert: Agent-Based Models in NetLogo (Quantitative Methods in Science) 2008

[6] Nick Bennett, Bob Robey, and Tom Robey :Computational Solution Techniques In Mathematical Programming

[7] Linge Bai and David Breen: Center of Mass in an Unbounded 2D Environment

[8] Wikipedia – genetic algorithms and steepest descent.

[9] Bob Robey, Tom Robey: Talk on optimization algorithms at Supercomputing Challenge Kickoff 2009.

[10] The NetLogo flocking model:
Wilensky, U. (1998). NetLogo Flocking model.
http://ccl.northwestern.edu/netlogo/models/Flocking.
Center for Connected Learning and Computer-Based Modeling,
Northwestern University, Evanston, IL.

[11] ParaView: www.paraview.org.

9.0 Acknowledgements

First and foremost, we would like to thank the people of the Supercomputing Challenge. This project has opened the eyes of everybody on this team about the depths and usefulness of computer programming. We would like to thank in particular Bob Robey for helping us formulate an idea for a project.

Secondly, team 70 would like to thank Mr. Goodwin, our sponsor teacher, as he has encouraged us through this difficult journey and kept us enthusiastic about the task at hand. He has provided a warm and comforting atmosphere for our team, and the school, to get together and move forward in our work.

Thirdly, we would all like to thank our sponsors, Christine and Jim Ahrens for their expertise in computing and data visualization.

Fourthly, we would like to thank all of the judges. The judges took time out of their busy lives to listen to our proposals and interim reports and give us feedback on what we need to work on. They gave us positive feedback as well as suggestions for the future to make our project the best that it could be.

And last but not least, we would all like to thank our families, our moms, dads, sisters, and brothers.