Exploring Genetic Algorithms Through the Iterative Prisoner's Dilemma

TJHSST Computer Systems Lab 2007-2008

Aaron Dufour

Iterative Prisoner's Dilemma

The general Prisoner's Dilemma is a scenario in which there are two players that can each choose to either cooperate with the other or to defect. They must each make their decisions without knowledge of the other's decision. Each player then gets points based on what their decisions were. The points are given as follows:

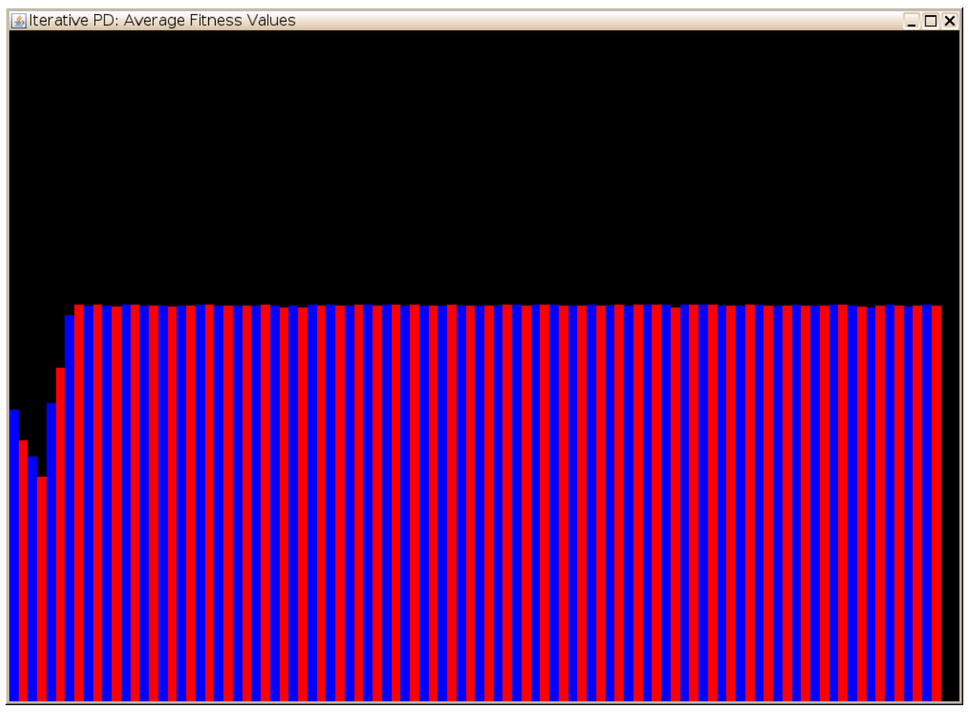
	Cooperate	Defect
Cooperate	R,R	S,T
Defect	T,S	P,P

The point values must satisfy certain inequalities. In order to be a Prisoner's Dilemma problem, the inequality $R > T > P > S$ must be satisfied.

In the Iterative Prisoner's Dilemma, the two players go through this scenario many times, and remember the past. In order to be used for the Iterative Prisoner's Dilemma, the values must satisfy the inequality $2R > S + T$. The points from each round are added to form a score for each player. I used the following table, which satisfies both inequalities and is the most commonly used set of values:

	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

The output of my program is a graph of the average fitness value for each generation (shown below) as well as the numbers represented by this graph in a text file, so that the graph can be recreated after the program is closed. I will use these graphs to determine how many generations the algorithm took to reach the best solution, and compare these among different algorithms.



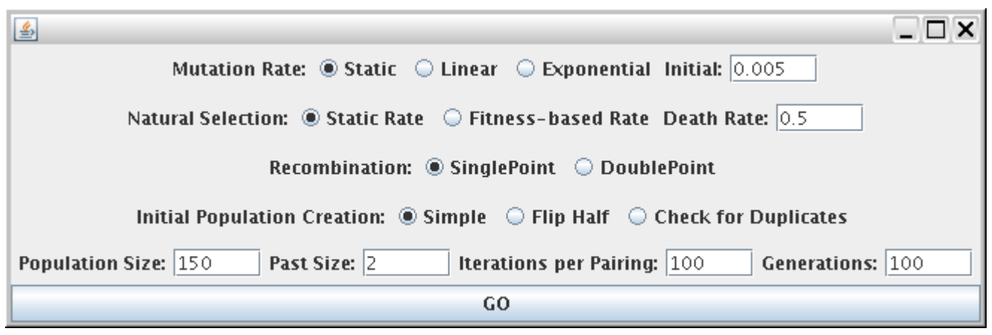
Another method takes the data that is shown in a graph such as the one above, and finds the point at which the fitness level stabilizes. It finds this point by eliminating data from the left until the slope of a fit line is clos enough to zero ("close enough" based on many test runs). This will allow data collection to be completely automated so that many runs of the genetic algorithm can be used to formulate conclusions about the different methods.

Genetic Algorithm

Genetic algorithms are used to find approximate solutions to optimization problems when the solution would be very time-consuming to compute. The general layout of a genetic algorithm is:

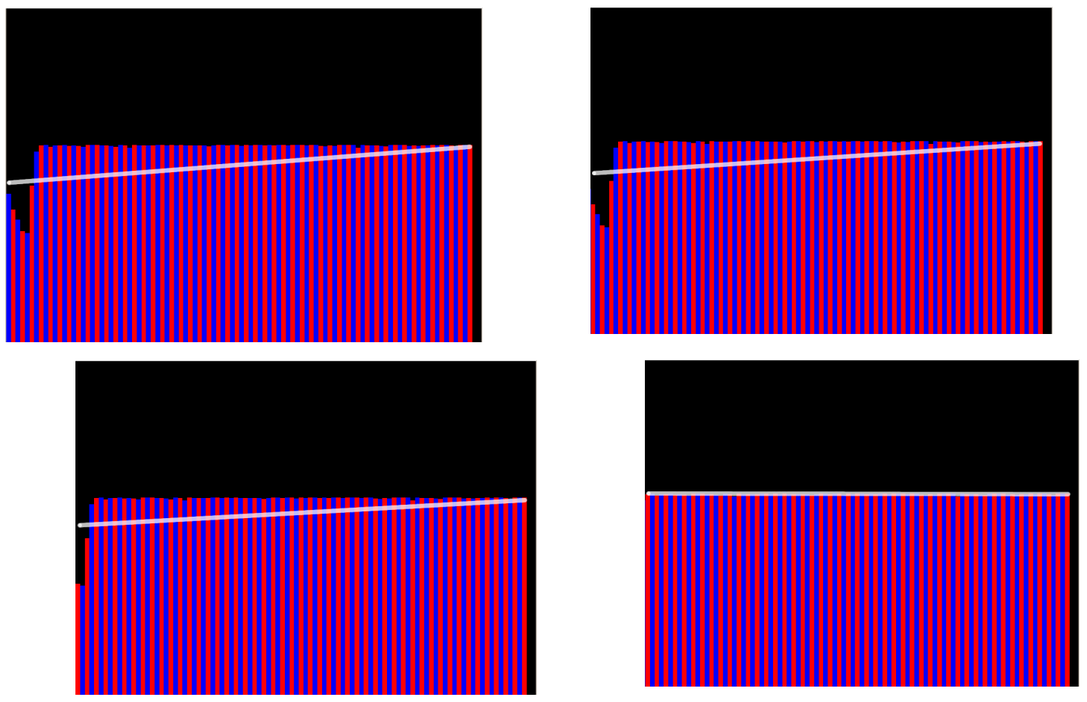
Initialization of gene pool
Loop over generations
    Natural Selection
    Selection
    Loop over empty slots in population
        Recombination
        Mutation

There are many ways in which to perform each of these steps. My program allows the user to select which method to use for each step. In this way, the different methods can be compared.



Determining When the Genetic Algorithm Finished

Although it is usually obvious to a person when the genetic algorithm finished based on the graph, the randomness from the mutations prevents a program from easily finding this information. The method that I developed to perform this uses the least-squares method to find the slope of the data. If the slope is not close enough to zero ("close enough" defined based on testing) it eliminates the left-most data point and does the same thing again. Once it finds a point where the slope is "close enough" to zero, it returns the number of iterations that it took to find that point.



Final Run

After ensuring that the genetic algorithm worked properly with all of the parameters, the next step was to run it with each of the parameters that I wanted to test. A series of nested loops automated this process, allowing the program to output all of the data in one run. The data was a file for each configuration, containing the number of iterations that each of the 10 runs with that configuration took, and an average of those 10 numbers.

Analysis

Analysis of the data from the final run showed that I could not make any conclusions about the mutation rate or the initial population creation because the population was too large and the number of possible solutions was very limited because I only allowed them to remember the past 2 generations, so which method was used had very little effect on the outcome. However, I did determine that the fitness-based natural selection outperformed the static natural selection, and that the double-point recombination slightly outperformed the single-point recombination.