Evolutionary Algorithm for Determining Stable Nash Equilibrium Strategies for Two-Person Bimatrix Games

Zac Schaefer
May 5, 2012

Abstract

This paper considers three relatively simple evolutionary algorithms for determining stable Nash equilibrium (NE) strategies for two-person bimatrix games. In general a bimatrix game may have more than one NE, and further analysis is required to determine which NE points are stable. That is to say, which NE would arise in a population of many players playing the game repeatedly? The algorithms discussed here provide a possible method to determine stable NE points.

Introduction and Background

A bimatrix game is specified by an $m \times n$ matrix of number pairs which indicate the payoffs to each player. An example 2×2 bimatrix would be

$$\begin{pmatrix} (1,2) & (-2,4) \\ (4,-3) & (1,0) \end{pmatrix}$$

We think of player I as the row player and player II as the column player. For example if I played row one and II played column two the payoff to player I would be -2 and the payoff to II would be 4. More generally players can choose to play a row or column with a certain probability by specifying a strategy vector. For instance I could choose the strategy vector $X = (.3, .7)$ and II could choose $Y = (.5, .5)$. So I would play row one 30% of the time and row two 70% of the time. Similarly player II would play either column with equal probability. Let A denote the matrix constructed from the first entries of the sample matrix above and let B denote the matrix constructed from the second entries.

$$A = \begin{pmatrix} 1 & -2 \\ 4 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 4 \\ -3 & 0 \end{pmatrix}$$

For the strategies listed above, the expected payoff to player I would be $XAY^T = 1.6$ where Y^T denotes the transpose of Y. Similarly the expected payoff to player II would be $XBY^T = -.15$. Each player would like to choose the strategy which

maximizes her payoff when played against the opposing player's strategy. If both players can find strategies such that neither player can do better by moving to a new strategy given that the opposing player's strategy remains fixed, then a NE point has been found.

Playing a particular NE strategy does not guarantee that both players are receiving the maximum possible payoff. Indeed if both players simultaneously switch to new strategies it is possible that each payoff could be increased. If there are several NE then some will give better payoffs than others. If we have a population of players playing the same game repeatedly, it is not always clear which NE is the most stable. In other words which, if any, NE strategies would persist if the game was played a large number of times and both players could adjust their strategies freely? For instance suppose both players start by using strategies given by a particular NE. A unilateral move by one player from the NE point may decrease her payoff at first, but on the next play the opposing player may adjust his strategy to account for the shift, resulting in a better payoff for both players. This may happen for several plays in a row resulting in a shift from the original NE point. In this case the NE point would be unstable.

The Algorithms

I consider an algorithm which is meant to evolve two populations of opposing players. Each of the three algorithms described below uses a different selection method to move from one generation to the next. The main steps executed in each algorithm are as follows:

1. Initialize two populations of opposing players. Randomly initialize all strategy vectors. Initialize the life of each player to 0.

2. Players from one population are then randomly paired with opposing players from the other. The expected value of the game is calculated using the strategy vector of each player and the game matrix. Next, the life of each player is adjusted according to the payoff obtained.

3. The previous step is repeated a fixed number of times.

4. The players are then ranked according to their life which consists of the sum of payoffs obtained in steps 2 and 3.

5. Players are selected according to the algorithm used and they produce offspring. The offspring are new players whose strategy vectors consist of a linear combination of the parents' strategy vectors.

6. Certain players (usually the worst performers) in the population are replaced by the offspring.

7. A mutation is applied to all players in the population which randomly adjusts each strategy vector by a small amount.

8. The life of each player is initialized to 0 and the process repeats.

The algorithm evolves the populations in a manner analogous to natural selection. After a sufficient number of iterations, the strategy vectors of each population should tend towards a stable NE.

After each iteration, the program calculates the average strategy vector of each population and uses this to calculate the average expected payoff to players 1 and 2. This average expected payoff is then printed to a text file. When the standard deviation of the last 20 average expected values falls below a certain number specified by the user, the mutation rate begins to decrease. This indicates that the algorithm is beginning to converge to a value and a smaller mutation rate allows the strategy vectors to converge more readily to the possible NE.

The first algorithm, labeled Update1 in the code, evolves each population by replacing a fixed number of the worst performing players with offspring from the best performing players. The players chosen to produce offspring are chosen randomly from the top k players where k is specified by the user. A mutation is then applied to the players in the population which were not replaced by offspring from the top players.

The next algorithm, labeled Update2 in the code, evolves each population using a geometric probability distribution to select which players will produce offspring. For example, if the parameter for the geometric distribution is .9, then the best player will be selected with probability .9, the second best with probability $(.9)(.1)$, and the nth best with probability $(.9)(.1)^n$. Since we do not have an infinite number of players, if the sample from the distribution falls below a certain number, the worst player is selected. In this way, we can give the best players a high probability of being selected, but some of the mediocre players will be selected too, maintaining higher diversity in the population. In Update1, a fixed number of poorly performing players are replaced each iteration. In contrast, Update2 replaces the entire population with the offspring of the selected players each iteration. The same mutation as in Update1 is then applied to all players of the population.

The last algorithm, labeled Update3 in the code, is similar to Update2 but uses a different technique to select which players will produce offspring. Here we select a random pool of n parents and then select the top two performing players from the pool to produce offspring. Like Update2, the entire population is replaced by

the offspring of the previous generation in each iteration. The same mutation as in Update1 and Update2 is then applied to all players of the population.

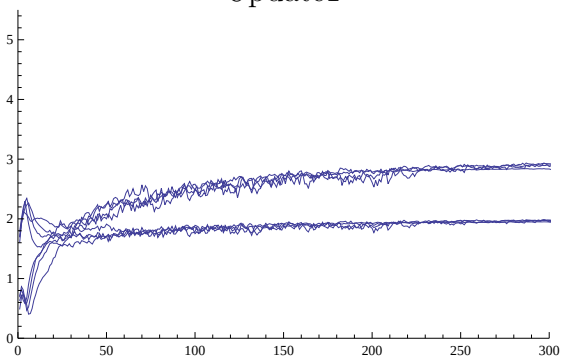
Example Run

As an example, consider the 3×3 bimatrix game

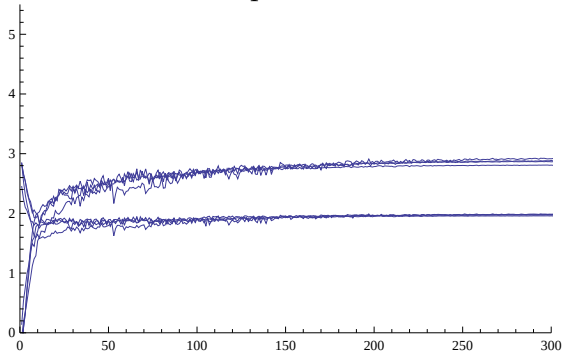
$$\begin{pmatrix} (-2,-4) & (5,-2) & (1,4) \\ (-3,-3) & (2,1) & (3,4) \\ (2,3) & (1,1) & (3,-1) \end{pmatrix}$$

The plots below represent the average game values for four runs of 350 iterations each for Update1, Update2, and Update3 respectively. The player 1 and 2 populations both consist of 35 players. During each iteration each player plays the bimatrix game against 25 randomly selected opposing players. The probability of a player experiencing a mutation is .25 and the level of the mutation is set at .1. The mutation level indicates a proportion which is randomly subtracted from one component of the strategy vector and added to another. Finally, if the standard deviation of the last 20 average game values falls below .06, the mutation level begins to decrease at a rate of $(.985)^k$ where k begins at 1 and increases by one each generation. In the first plot the bottom 5 players are replaced each generation with offspring from the top 2 players. In the second plot the parameter for the geometric distribution is .9. The third plot uses a pool of size 5 from which to select parents for the new offspring.

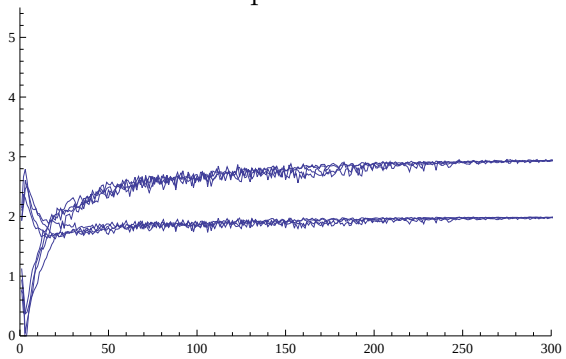
Update1



Update2



Update3



Three possible NE points exist for this game. The game values or payoffs to each player at these points are $\{(3, 1.007), (2, 3), (2.21, .78)\}$. The exact terminal payoffs from the plots above are given in the table below along with the corresponding average strategies and the average running time in seconds of each algorithm.

Algorithm	Payoff 1	Payoff 2	Strategy 1	Strategy 2	Time
Update1	1.98	2.93	(.005,.004,.991)	(.956, .03, .014)	7.93
Update2	1.96	2.93	(.003,.001,.996)	(.955, .03, .015)	9.8
Update3	1.99	2.96	(.001,.002,.997)	(.985, .009, .006)	13.12

From these results it is clear that the populations are tending toward the NE point at $\{(2, 3)\}$ with the pure strategy (0,0,1) for player 1 and (1,0,0) for player 2. Therefore, it is reasonable to assume that the NE point at $\{(2, 3)\}$ is stable and would always arise if this game were played repeatedly.

We can see that all three algorithms give relatively fast convergence. Generally, a population in the range of 30-50 seems to work well for most problems. A mutation probability in the range .15-.30 and a mutation level in the range .08-.15 usually works well for this population size.

Conclusion

In summary, these algorithms give a possible alternative to analytical techniques for determining stable NE. Ideally, a number of NE points can be calculated (using the Lemke-Howson algorithm for example) and these algorithms can be used to decipher which NE are stable. If no NE points are known, the algorithms may be used to obtain estimates for possible stable NE points. In many cases the parameters such as the mutation probability and parent selection method will need to be adjusted for each problem. The advantage of these algorithms lies in their relative simplicity and the fact that they can be used on any $n \times n$ bimatrix game. However, there are disadvantages as well, not least of which is the fact that it may be difficult to adjust the settings correctly if no NE points are known from the outset. As with most nondeterministic algorithms, it is also possible to have premature convergence if the tolerance threshold for decreasing the mutation level is set incorrectly. Regardless of these shortcomings, the algorithms do have the desirable property of simulating the interaction which might actually take place between two opposing populations of players.