Genetic Algorithm - Prisoners Dilemma

Marek Skrzypecki, INF, sem. 6, GKiO2

Introduction - The Game

1. Two players.
2. Each player can either “Cooperate” (C) or “Defect” (D) .
3. The score received by the players depends on the move made by both of them.

Playoff Matrix

		Player 2	
	Decision	Cooperate	Defect
Player 1	Cooperate	R=3 R=3	S=0 T=5
	Defect	T=5 S=0	P=1 P=1

R: Reward, S: Sucker, T: Temptation, P: Penalty

$T > R > P > S$, $(T + S)/2 < R$

ITERATED PRISONER'S DILEMMA

The situation is more interesting when the players play the game iteratively for a certain number of moves.

- The number of moves should not be known to the two players.
- The winner is the player with the highest score in the end.
- Non zero Sum Game - Both the players can simultaneously win or lose!
- No Universal Best Strategy - The optimal strategy depends upon the opponent

Why use Genetic Algorithm?

- Total number of all possible strategies is very high - 2^70. Exhaustive search will take more than a lifetime!
- Fitness function: Non continuous, classical methods will not work
- Genetic Algorithms emulate the natural process of evolution.

Solution - Chromosome

Chromosome is represented as 70-letter long string
e.g. CCDCDDDCDCCCCDDCDCCDDCDDC…
where 64 letters encode a strategy (every letter is a decision for a different possibility, based on three previous moves) and 6 letters encode three hypothetical previous moves used by the strategy to decide how to move in the first actual game.

Since each locus in the string has two possible alleles (C and D), the number of possible strategies is 2^70, mentioned before.

Solution - Fitness Function

The Prisoner's Dilemma provides a natural means of evaluating the success, or fitness, of each solution – the game payoffs. We can state that the strategy, which earns the highest payoff score according to the rules of the Iterated Prisoner's Dilemma, is the fittest, while the lowest scoring strategy is the weakest.

However these 'raw' fitness values present some problems. The initial populations are likely to have a small number of very high scoring individuals in a population that will take it over rapidly and cause the population to converge on one strategy.

Solution - Fitness Scaling

It is useful to scale the ‘raw’ fitness scores to help avoid the above situations. This algorithm uses linear scaling that produces a linear relationship between raw fitness - f and scaled fitness f’ as follows:

$$f' = af + b,$$

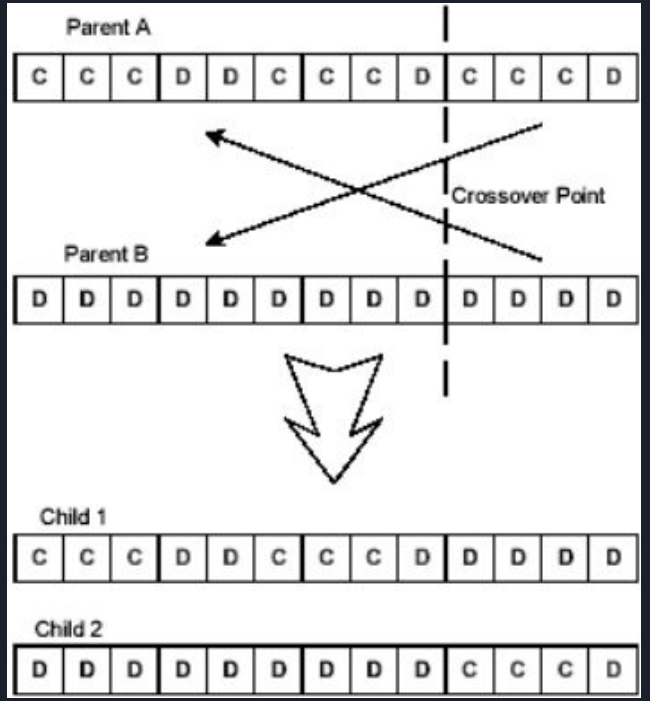
Coefficients a and b may be calculated as follows:

$$a = (c - 1) * \frac{f_{avg}}{f_{max} - f_{avg}} \qquad b = f_{avg} * \frac{(f_{avg} - (c * f_{avg}))}{f_{max} - f_{avg}}$$

Where c is the number of times the fittest individual should be allowed to reproduce. A value of 2 was found to produce accurate scaling in this method.

Solution - Crossover

This algorithm breaks both the parent chromosomes at the same randomly chosen point and then rejoins the parts, producing children.



Solution - Mutation

Mutation will have a small possibility of occurring (0.1%) and will consist of a bit copied between the parent and the child being flipped (One letter in the chromosome string changed from C to D or D to C).

These mutations provide a means of exploration to the search.

Replacement

The genetic algorithm is run across the population until it has produced enough children to build a new generation. The children then replace all of the original population.

Planned Technology

Implementation of the algorithm in C ++.

Work Plan

1. Preparation of a class representing chromosomes.
2. Implementation of Fitness Function and Fitness Scaling
3. Preparation of mutation and crossover operators.
4. Implementation of working algorithm using the prepared components.
5. Testing the program.

Thank you!

Author:
Marek Skrzypecki