AI Lab 4 part 2

CoEvolution

Submitted BY :

Abed Abo Hussien 208517631

Samer Kharaoba 209050202

First of all the changes done to the algorithm are as follows:

1. Mutualism function
2. Communalism function
3. Parasitism function

We used all three to create the new algorithm, by replacing the cross function in the previous GA with all three stages.

1. Each stage takes a gene and manipulates every single item in that gene:
2. Each gene from the population is comprised of a string with length 1000
3. The implementation: we took each item in each string as a vector X and used the equations that we learned in the lecture to change them

* For example :
* Then applied each equation on each set

Now lets explain each stage :

Mutualism function:

* It was implemented as described in the lecture and we already explained how we view each “vector” as stated above
* Select a random ,get random Benefit factor ,Create Mutual vectors ,use all of the above in the equation
* Then update the

Text

Description automatically generated

Communalism function

* Exactly as above function but now update only

Text

Description automatically generated

* The thing to note about Communalism and Mutualism is that they replace the cross functionality

Parasitism

* **The nice thing about this function Is that it mutates a section of the population , and replaces the mutate functionality , but it is more aggressive than the mutation function of the original GA.**
* **Exactly the same parameters as the previous functions but adds the mutation probability**
* **Parasite vector is calculated based on the mutation probability**
* **We can use adaptive decrees to change the ratio of exploration vs exploitation**

**Text

Description automatically generated**

We also created a match between our agent and all dummies and experts as a guideline for the fitness:

1. For every citizen ,check how many matches it **doesn’t lose** and we put that as the fitness
2. Granted this fitness changes from match to match, but it also means that we get competitive benchmarks in between the population
3. And a function called winner that determines the winner of the game

Text

Description automatically generated

The function called fit\_dummies established 1000 matches between our agent and the rest of the robots and calculates the fitness as the number of wins against all robots divided by the number of robots :

Text

Description automatically generated

The function fitness uses above function on all agents in the population affectively calculating fitness for all of them :

Text

Description automatically generated

Mate :

1. selects elite society
2. uses cross that we changed as 1.mutalism 2. communalism 3.parasitism

Text

Description automatically generated

Note: this algorithm changes the population dynamically and doesn’t put them in a buffer

By now we explained the full algorithm of coevolution

Now we need to show the actual match between all participants:

We created a roshambo match class :

1. start match creates a match between an agent vs dummies and experts
2. match\_X\_Y: creates a match between members of X and members of Y
3. all\_out\_war : creates a match between all participants

Text

Description automatically generated

Sections to answer :

Section 1,2,3,4 :

|  |  |  |  |
| --- | --- | --- | --- |
| Agent | Section1:  Deterministic/Stochastic | Section 2/4: 1. Learn/predict  2. Meta Strategic/Random/Set of moves /Mix | Section 3: Exploration/Exploitation |
| Anti Flat Player | Deterministic | 1. Prediction | Exploitation |
| Copy Player | Deterministic : always plays according to the last move made by the opponent | 1. learns : opponents last move  2.Uses Meta strategy | Exploitation |
| Freq Player | Deterministic : checks the most frequent move of the opponent and plays the winning move against it | 1. Prediction: tries to predict next move based on former moves  2.learns: the most frequent move | Mix of both |
| Flat Player | Stochastic | 1.doesn’t learn or predict  2.Random : uses a coin flip to determine next move | Mix of both |
| Foxtrot Player | Stochastic | 1.doesn’t learn or predict  2.Random : uses a coin flip to determine next move | fairly Explorative |
| Bruijn 81 Player | Deterministic :uses constant set of moves | 1. doesn’t learn or predict  2.Constant set of moves | Neither |
| Pi Player | Deterministic : Pi is constant thus the actions are deterministic | 1. doesn’t learn or predict  2.Can be reduced to a constant set of moves | Partially exploitive |
| 226 Player | Stochastic : is based on 20% 20% 60% distribution | 1. doesn’t learn or predict  2.Mix of strategies : uses Randomness with preset probabilities | Mix of both |
| Random Player | Stochastic: uses randomness to make moves | 1. doesn’t learn or predict  2.Random: uses randomness to make moves | Partially explorative |
| Rotating Player | Deterministic : it follow a pattern and knows what move to make based on that pattern | 1. predict’s the next move from the simple strategy that it follows  2.uses a strategy but not a meta one | Partially explorative |
| Switching Player | Stochastic | 1. Learns from past moves made by the opponent 2. Uses Randomness | Exploitation |
| Switch a Lot Player | Stochastic : works like Switching Player but sometimes uses it’s previous course of action | 1. Learns from past moves : as it might use a past move that it made 2. Uses Randomness | Exploitation |
| Greenberg | Stochastic : uses Randomness | 1. Learns from past moves and predicts next move 2. Uses Randomness | Both as it learns and predicts |
| Iocaine | Deterministic | 1. Learns from past moves 2. Uses meta-strategy | Both ,same reason as above |

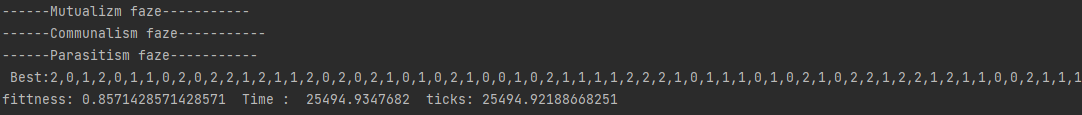
Section 5 :

As we stated before , our population is comprised of Genes , each Gene contains an array of size 1000 of randomly created moves , we added a new Class named RPS that uses agent as a father Class , we changed the character creation function so that each element of the array is comprised of either 0,1,2 respectively :

Text

Description automatically generated

The resulting agent looks something like this (after it gets through all the stages in the CO-Evolution-GA ):



Section 6+7:

We already explained in the first couple of pages how the flow of the algorithm is and how the fitness function works

Note: The stoppage criteria of the algorithm is that fitness =1 or max iterations reached

Section 8:

How we couped with the resulting problems associated with co-evolution algorithm:

1.we used adaptive decrease to allow for gradual exploration-exploitation ratios which helped with the three given problems

2. we made sure that in each stage of the algorithm to change the most fit Gene only when we find a better suiter to the position witch helped with the circularity problem .

Section 9:

Our agent :

|  |  |  |  |
| --- | --- | --- | --- |
| Agent | Section1:  Deterministic/Stochastic | Section 2/4: 1. Learn/predict  2. Meta Strategic/Random/Set of moves /Mix | Section 3: Exploration/Exploitation |
| Our agent | Deterministic : it’s true that we used randomness to create our agent but after the evolution process it sums up to be a constant set | 1. doesn’t predict nor use a meta-strategy  2. Static set of moves | None |

Section 10:

In the folder output you can find all the relevant results of all simulations done by us ,one of them is the following results :

We recorded the results of 20 matches (in the code you can choose the number )

The numbers in the last row are the averages of all above values   
We tried using evolution with fitness calculated as :

1. Tournament with all Agents (took too much time but we did it just for testing)
2. Tournament only with Dummies

Results of Tournament with all Agents as fitness function:

A picture containing text, light

Description automatically generated

Chart

Description automatically generatedTable, calendar

Description automatically generated

Summary of above values :

|  |  |
| --- | --- |
| Opponent | Agent Wins on average: |
| Anti Flat Player | 100% |
| Copy Player | 100% |
| Freq Player | 100% |
| Flat Player | 55% |
| Foxtrot Player | 50% |
| Bruijn 81 Player | 100% |
| Pi Player | 100% |
| 226 Player | 15% |
| Random Player | 35% |
| Rotating Player | 100% |
| Switching Player | 60% |
| Switch a Lot Player | 55% |
| Greenberg | 35% |
| Iocaine | 45% |

And its average win ratio is 67% i.e wins against 67% of it’s opponents

Results of Tournament only with Dummies (2) :

Table

Description automatically generated with low confidence

Chart

Description automatically generatedTable

Description automatically generated

|  |  |
| --- | --- |
| Opponent | Agent Wins on average: |
| Anti Flat Player | 100% |
| Copy Player | 100% |
| Freq Player | 100% |
| Flat Player | 55% |
| Foxtrot Player | 55% |
| Bruijn 81 Player | 100% |
| Pi Player | 100% |
| 226 Player | 45% |
| Random Player | 55% |
| Rotating Player | 100% |
| Switching Player | 70% |
| Switch a Lot Player | 50% |
| Greenberg | 40% |
| Iocaine | 45% |

|  |
| --- |
| Previous margins |
| 100% |
| 100% |
| 100% |
| 55% |
| 50% |
| 100% |
| 100% |
| 15% |
| 35% |
| 100% |
| 60% |
| 55% |
| 35% |
| 45% |

Summary of above values:

Wins on average 72% of the matches it plays and we have a 5% improvement over the evolution with all agents   
moreover it improved against several opponent’s one huge jump would be against 226 player and random player by 20 to 30 percent improvement on those to players

Note: we do think that if Greenberg and iocaine were faster we would be able to get better margins using the first approach as we would then increase population size and then get more diversity in the solution

Section 11:

First approach

The last tournament (All out war as we called it ) was done 5 times so that we understand the behavior of our agent , we will post the last round here , the specific result is in “outputs\20\_iter\_50\_popSize\_20Rnds\res out”

As well as the previous results all are found in “outputs\20\_iter\_50\_popSize\_20Rnds”

Text, letter

Description automatically generated

Text

Description automatically generated

Results:

if we arrange the win ratio of all agents :  
1. Iocain

2.Our Agent

3. Greenberg

4.copy ,freq,266

5. all of the rest

Thus showing that our agent comes at second place after iocain which is an impressive result .

If we look at the full folder that contains the results we can see that in 5 grand tournaments our agent comes mostly second behind Iocain and in the rest of the cases comes 3rd behind iocain and Greenberg

For a deterministic Agent it performed amazingly

Second strategy :

Under file output with the name try\_functionality folder :

Text, letter

Description automatically generated

One of the runs :

**Text

Description automatically generated**

Results:

if we arrange the win ratio of all agents :  
1. Iocain

2.Our Agent

3. The rest of the agents

Thus showing that our agent comes at second place after iocain which is an impressive result .

If we look at the full folder that contains the results we can see that in 5 grand tournaments our agent comes mostly second behind Iocain and in only one case came 3rd behind iocain and Greenberg

For a deterministic Agent it performed amazingly

How to operate :

simply run the exe file and enter what is asked of you

Things to improve:

1. Create a meta-strategy to help improve our results , we wanted to use Markov chains , but couldn’t due to time constraints ,and personal issues
2. Try to find a solution that isn’t deterministic because one string of moves can’t trump all possible agent’s

Please note that there are more tests in the output file than what is shown here