**Round Robin and Evolutionary Tournament - Based Comparison of Strategies for Repeated Normal Form Games**

Areg Karapetyan

CIS, Masdar Institute of Science and Technology

Abu Dhabi, UAE

akarapetyan@masdar.ac.ae

Abstract

Evolutionary tournaments have been used as a tool for comparing strategies. For instance, in the late 1970’s, Axelrod organized tournaments to compare strategies for playing the iterated prisoner’s dilemma (PD) game. While these tournaments and later research have provided a better understanding of successful strategies for iterated PD, the understanding is less clear about strategies for playing iterated versions of different games. In addition it is worthy to mention that game theorists are still in a need for a common tool (application) to conduct complex tournaments. Thorough research in web recourses clearly shows that PD (and not only) tournaments are of great popularity. For instance, [www.brilliant.org](http://www.brilliant.org) conducts PD tournaments on a year basis and the winner gets valuable reward. In this report, I discuss the importance of the need for creating a uniform open source application for conducting round robin and evolutionary tournaments. Also, report includes performance comparison and analysis of the “Siri” strategy.

Keywords—round-robin tournament, evolutionary tournament, normal form games, matrix games,

# Introduction

|  |  |
| --- | --- |
| No. | Strategy name |
| 1 | Maximin |
| 2 | Always Defect |
| 3 | Win-Stay-Lose-Shift |
| 4 | Tit-For-2-Tats |
| 5 | Siri |
| 6 | Random |
| 7 | Always Cooperate |
| 8 | Tit-For-Tat |

This report introduces comparison of strategies mentioned in the Table1 based on round robin and evolutionary tournaments and detailed analysis of the tournament’s results. Also, it introduces new algorithm “Siri” and discusses its performance and effectiveness in the tournament. In order to conduct and analyze the tournament I have created an application “Hungry Games”, which has been developed using Python, JQuery programming and HTML, CSS markup languages. With the help of that application a user can choose any combination among given games and strategies and conduct a round robin tournament with the chosen number of rounds. Also, based on the results of the tournament application automatically generates and shows players average payoff matrix and evolutionary tournament graph.

Code of the application is available on Github1 and the standalone executable version for Windows and Mac OS can be obtained upon request.

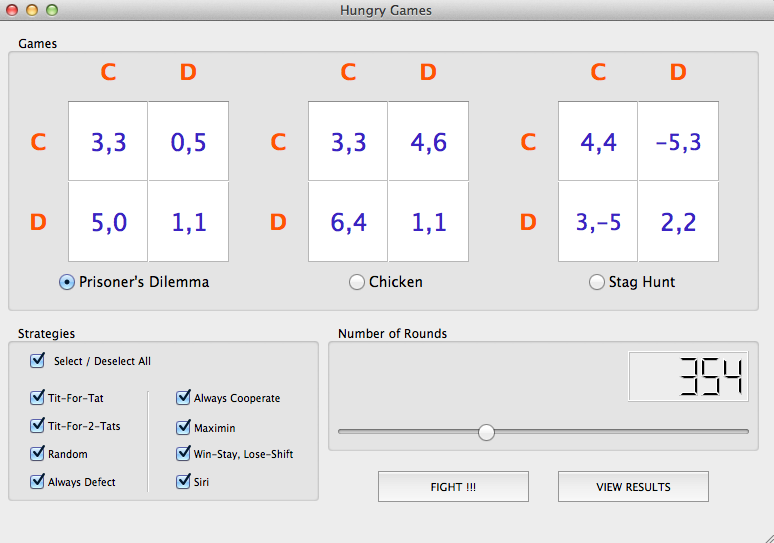
# Tournament Details and requirements

This report includes detailed analysis and description of PD, Chicken and Stag Hunt round robin and evolutionary tournaments. Each of the round robin tournaments’ specifications is as follows

1. Tournament should be conducted on all the algorithms mentioned in Table 1.

2. Each algorithm should play against itself and each of the other algorithms in a repeated game of 1000 rounds

It should be noticed that tournaments’ results would be more reliable if the number and type of strategies in Table 1 will be extended. For instance, it could contain “Never Forgive”, “ Fictitious Play “, “ Saby “ etc. [1]



**Fig. 1**

# Description of The algorithm “Siri”

There are different approaches in Game Theory in terms of normal form algorithms classification. Some of them classify algorithms into “ trigger ” and “ non-trigger ” strategies, others organize as “learning” and “non-learning” [1]. Undoubtedly there isn’t a single strategy, which will win tournaments on every game. On the other hand, there should exist such a concept, which could always succeed. Based on the analysis of Axelrod’s tournament it can be inferred that characteristics of the winning strategy “Tit For Tat”, which is a trigger strategy, are as follows

1. Nice

2. Forgiveness

3. Clarity

4. Reciprocate

“Trigger” strategies have 2 basic components: “offer” and “threat”. They propose an offer to opponent and as long as it is accepted trigger strategy plays an offer action. However, once the opponent deviates from offer trigger strategy plays the threat action. However, drawback of this approach is that it doesn’t take into account opponent’s performance during the all the previous rounds. In other words, it doesn’t consider opponent’s reputation. Playing this strategy will not always guarantee success. As a prove, “Tit For Tat” failed to succeed in all three tournaments.

On the other hand, strategies that observe opponent’s performance are called “ learning ” strategies. They observe and analyze opponent’s actions, try to predict its next step and play the best response to it.

There is a widely accepted notion that the strategy, which will be able to properly predict opponent’s next step and give a best response to it will the best strategy among all existing ones. Nevertheless, to the best of my knowledge such a strategy hasn’t been introduced yet. Anyway, it should be assumed that in order to succeed in a tournament using a learning strategy you should have naxnakan knowledge of analysis of the game and classification list of all possible opponents.

Considering abovementioned facts and observations, I have developed a different algorithm for each tournament. Below are presented those codes in Python programming language. However, the underlying concept of all of them is as follows

1. Start playing by triggering game specific pattern of actions/action
2. Keep learning the opponent for a game specific period

Until the end of the game keep

3. Observing the opponent’s actions

4. Classifying opponent’s reputation based on the observation

5. Play game specific best response based on classification result

Underlying concept of the “Siri” is to trigger the opponent, look for the response, based on the response find out its reputation and act against it based on its reputation. The main challenge of the strategy is to find a precise algorithm for reputation identification. Once the algorithm will identify opponent’s reputation it can achieve desirable results.

Performance analysis of the “Siri” could be explained with a statement

1. Observe and learn the opponent
2. Find out opponent’s reputation
3. Try to make the opponent a kingmaker if possible
4. Otherwise act with him on a equal basis

In other words, it can be stated as “Learn, Adapt, Win !”

In spite of the fact that “Siri” won all tournaments, it needs to be enhanced and modified. Hence I have had time constraints these are subjects for the future work.

***Algorithm of the “ Siri ” for Stag Hunt***

* def Siri(self, turn, previousrounds, place, game, lastscore):
* #I am observing other participants my Master
* QtyDefectionsOpponent = 0
* for i in range(0, len(previousrounds), 1):
* if previousrounds[i][1-place] == 'd':
* QtyDefectionsOpponent += 1
* if turn == 0:
* step = 'c'
* else:
* if QtyDefectionsOpponent == turn:
* #My Master, this oppponent is a defecter
* step = 'd'
* elif QtyDefectionsOpponent == 0:
* #My Master, this opponent is a cooperator
* step = 'c'
* else:
* step = 'd'
* return step

***Algorithm of the “ Siri ” for Prisoner’s Dilemma***

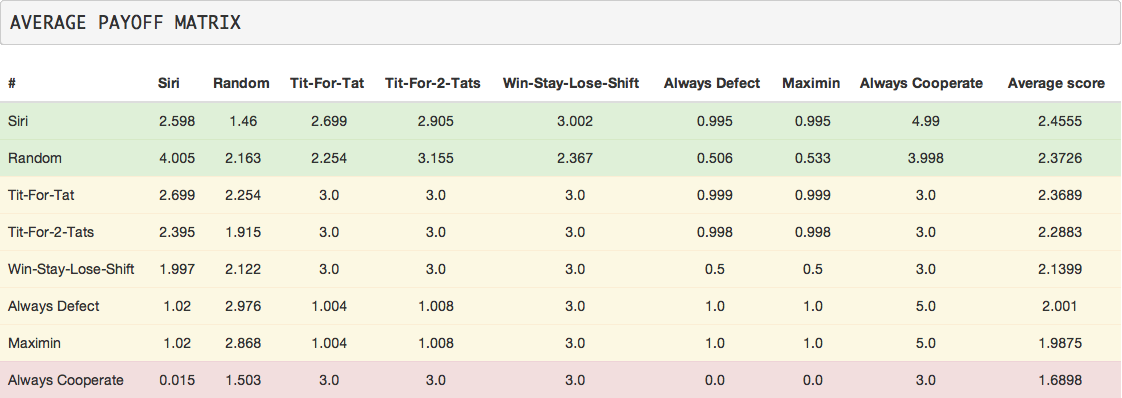
* def Siri(self, turn, previousrounds, place, game, lastscore):
* #I am observing other participants my Master
* initialsteps = ['c', 'd', 'c', 'd', 'c', 'd', 'c', 'c', 'd', 'c']
* QtyDefectionsOpponent = 0
* for i in range(0, len(previousrounds), 1):
* if previousrounds[i][1-place] == 'd':
* QtyDefectionsOpponent += 1
* if turn < 10:
* step = initialsteps[turn]
* else:
* if QtyDefectionsOpponent == turn:
* #My Master, this oppponent is a defecter
* step = 'd'
* elif QtyDefectionsOpponent == 0:
* #My Master, this opponent is a cooperator
* step = 'd'
* elif float(QtyDefectionsOpponent)/float(turn + 1) <= float(1)/float(5):
* #My Master, this is a kind opponent
* step = 'd'
* elif float(QtyDefectionsOpponent)/float(turn + 1) <= float(3)/float(5):
* #My Master, this is opponent most likely is similar to us
* step = 'c'
* elif float(QtyDefectionsOpponent)/float(turn + 1) > float(3)/float(5):
* #My Master, This is opponent most likely is a defecter
* step = 'd'
* return step

***Algorithm of the “ Siri ” for Chicken***

* def Siri(self, turn, previousrounds, place, game, lastscore):
* #I am observing other participants my Master
* initialsteps = ['c', 'd', 'c', 'd', 'c', 'd', 'c', 'c', 'd', 'c']
* QtyDefectionsOpponent = 0
* for i in range(0, len(previousrounds), 1):
* if previousrounds[i][1-place] == 'd':
* QtyDefectionsOpponent += 1
* if turn < 10:
* step = initialsteps[turn]
* else:
* if QtyDefectionsOpponent == turn:
* #My Master, this oppponent is a defecter
* step = 'c'
* elif QtyDefectionsOpponent == 0:
* #My Master, this opponent is a cooperator
* step = 'd'
* elif float(QtyDefectionsOpponent)/float(turn + 1) <= float(1)/float(5):
* #My Master, this is a kind opponent
* step = 'd'
* elif float(QtyDefectionsOpponent)/float(turn + 1) <= float(3)/float(5):
* #My Master, this is opponent most likely is similar to us
* step = 'd'
* elif float(QtyDefectionsOpponent)/float(turn + 1) > float(3)/float(5):
* #My Master, This is opponent most likely is a defecter
* step = 'c'
* return step

# Prisoner’s dilemma tournament

Player’s average score matrix for the round robin tournament is shown in Fig.2. Winner of the tournament is the “Siri” strategy. “Tit-For-Tat” and “Random” strategies are accordingly in second and third places. Evolutionary tournament graph based on the round robin tournament’s results is shown in Fig. 3.



**Fig. 2**

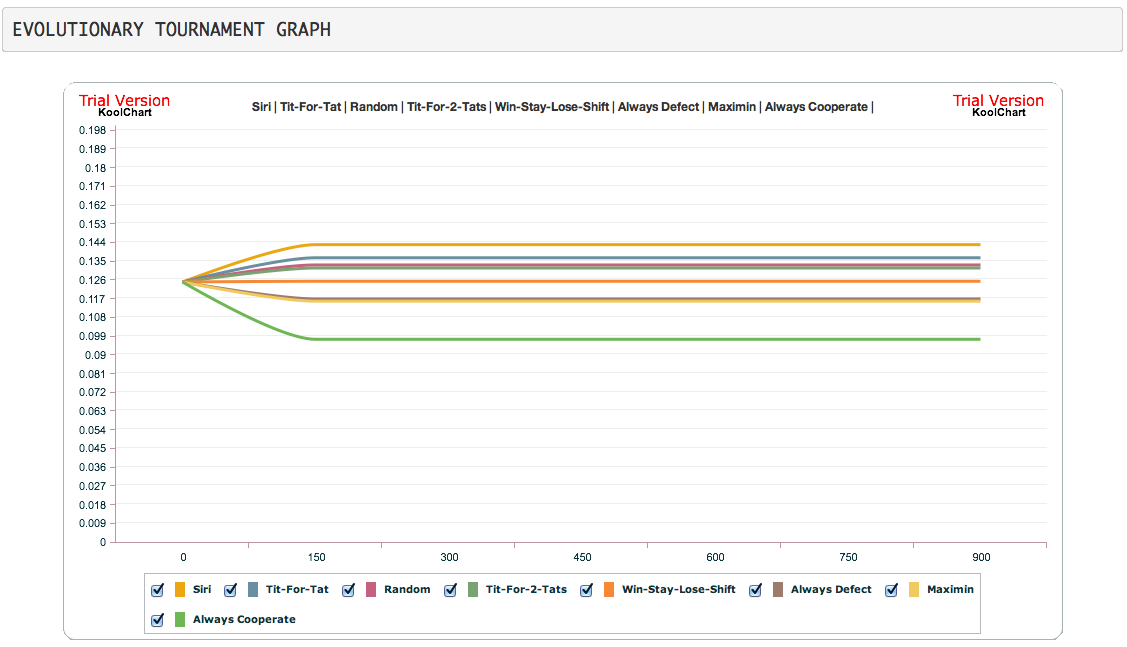
In this chapter I will provide detailed analysis of this tournament’s results. Following chapters will contain relatively short analysis. The facts that earn a success in the tournament are as follows

1. Find the weak strategies and make them kingmakers

2. Identify risky algorithms and properly act against them

3. Identify potentially dangerous algorithms and properly act against them

Fig. 2 shows that the kingmaker for the “Siri” was “ Always Cooperate ”. Siri successfully identified that “Always Cooperate ” will not defect against him and keep defecting against it. On the other hand, “Siri” did well with “Tit-For-Tat”, “Tit-For-2-Tats”, “Win-Stay-Lose-Shift” and against itself, because it was able to identify that if it defects against them then will get the same response from them, therefore it keeps cooperating with them. Also, “Siri” was able to identify potentially dangerous algorithms “Always Defect” and “Maximin” and act properly against them.



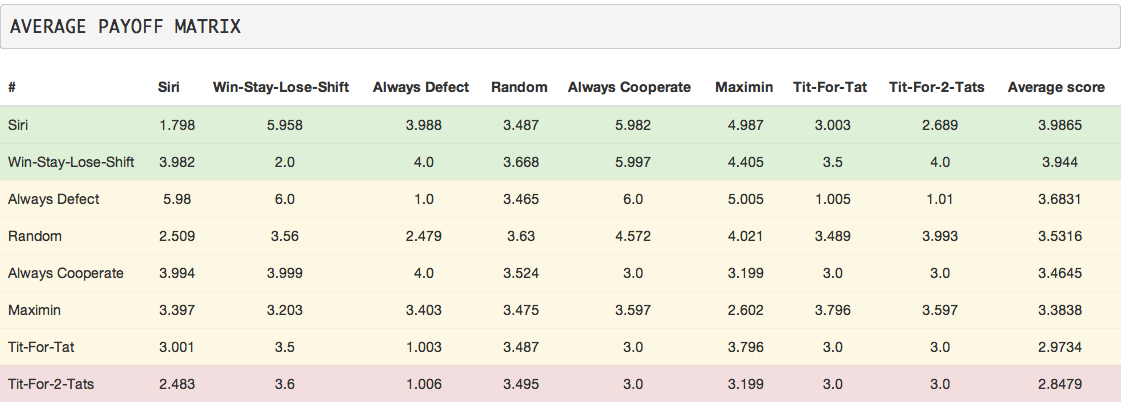
**Fig. 3**

The only strategy that “Siri” fails to identify correctly is the “Random” strategy. “Always Cooperate ” was the most unsuccessful strategy in this tournament. The main reason is that is

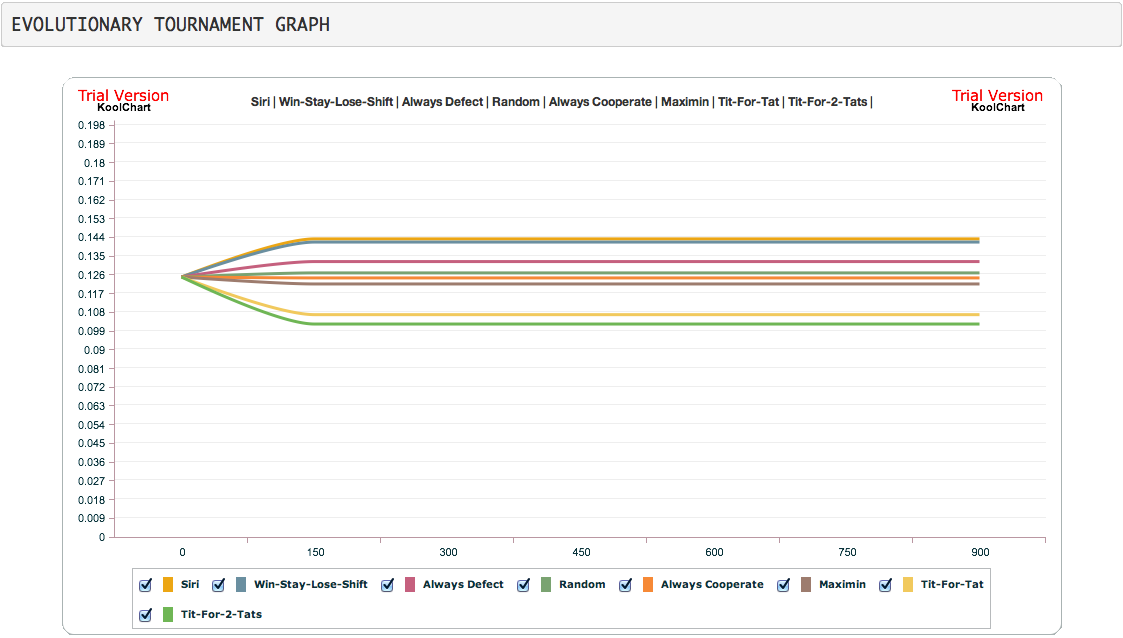
“Siri” also won the evolutionary tournament.

# Chicken tournament

Round robin tournament player’s average score matrix is shown in Fig.4. Winner of the tournament is the “Siri”. “Tit-For-Tat” and “Random” strategies are accordingly in second and third places. Evolutionary tournament graph based on the round robin tournament’s results is shown in Fig. 5.



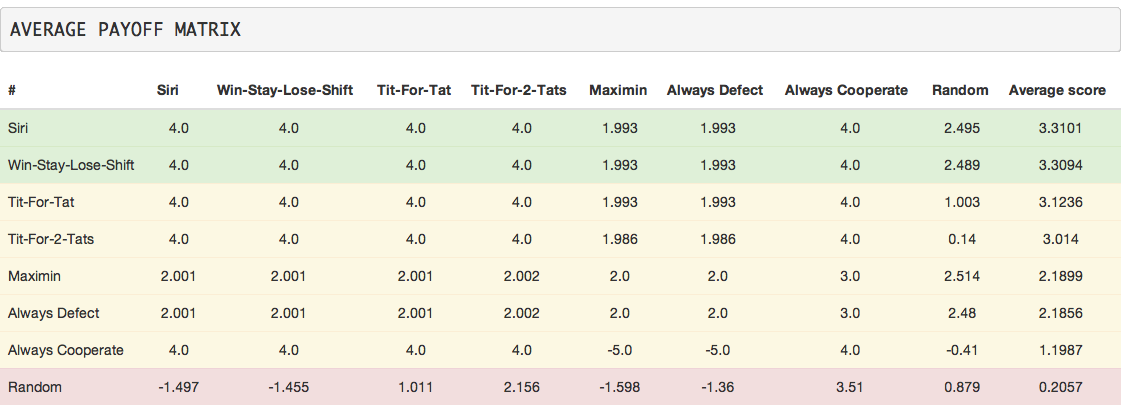
**Fig. 4**

****

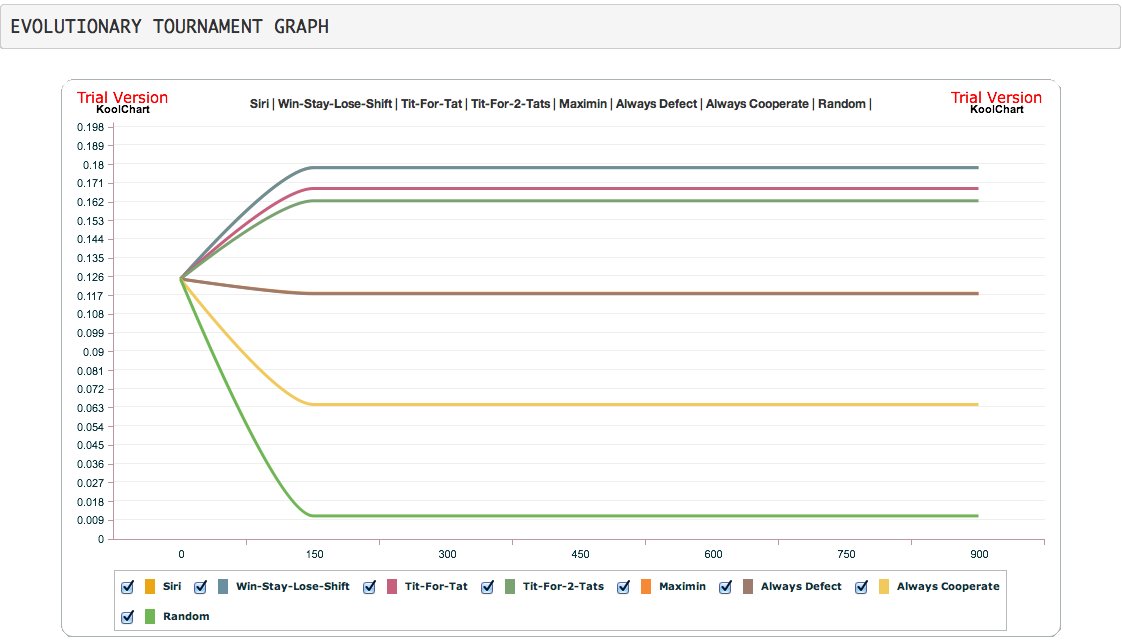
**Fig. 5**

# Stag hunt tournament

Round robin tournament player’s average score matrix is shown in Fig.6. Winner of the tournament is the “Siri”. “Tit-For-Tat” and “Random” strategies are accordingly in second and third places. Evolutionary tournament graph based on the round robin tournament’s results is shown in Fig. 7.



**Fig. 6**



**Fig. 7**

# Conclusion and Future work

All in all, game theorists are still looking for an algorithm that would be able to learn the opponent’s behavior, identify its reputation, predict its next step and give the best response to it. In fact, there still doesn’t exist a commonly used application for conducting and analyzing complex tournaments. Hence, I propose to enhance “Hungry Games” application’s functionality by creating more complex executable and web-based applications, which will allow to add/modify new/existing games/strategies dynamically. From my perspective, that will be a valuable tool for game theorists. In addition, enhance and improve the algorithm of the “Siri” in order to make it more effective.

# Bibliography

[1] - Ste ́phane Airiau, S. S. (2007/06/30). *Evolutionary Tournament-Based Comparison of Learning and Non-Learning Algorithms for Iterated Games* (Vol. 10). Journal of Artificial Societies and Social Simulation.