CITS4403
Computational Modelling

Week 11 Lecture: Evolution of Coorperation

Dr. Siwen Luo

Semester 2, 2024
School of Computer Science,
University of Western Australia

0 LECTURE PLAN

THE UNIVERSITY OF WESTERN AUSTRALIA

W11 Lecture: Evolution of Cooperation

1. Recap
2. Prisoner's Dilemma
   - Rational agents
   - Problem of Altruism
   - Prisoner's Dilemma tournaments
3. Simulating evolution of cooperation

1 Recap

Genotype: the genetic information of each agent that to be copied when the agent replicates.

A genotype can be represented by a sequence of N binary digits (0 or 1), where N is a parameter that we can choose.

Fitness: a quantity related to the ability of an agent to survive or reproduce.

Fitness landscape: a function that maps genotype to fitness.

This fitness is the "height" of the landscape. Genotypes which are similar are said to be "close" to each other, while those that are very different are "far" from each other.

Each genotype corresponds to a location on the fitness landscape.

Prisoner's Dilemma

Game Theory

Games represent interactions where there is a choice to 'cooperate' with each other or 'defect', and where the reward/punishment depends on what the 'opponent' chooses.

Contrived scenarios but they are still very useful to shed light on human motivation and behavior and have been the focus of extensive experimental research in all sorts of areas from social psychology to pure math.

One particular example of game theory, Prisoner's Dilemma, was first discussed in the 1950s.

Prisoner's Dilemma

Prisoner's Dilemma

Two members of a criminal gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack evidence to convict the pair on the principal charge, but they have enough to convict both on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain.

Each prisoner is given the opportunity to either: (1) betray the other by testifying that the other committed the crime, or (2) cooperate with the other by remaining silent.

- **If A and B each betray the other, each of them serves 2 years in prison.**
- **If A betrays B but B remains silent, A will be set free, and B will serve 3 years in prison (and vice versa).**
- **If A and B both remain silent, both will only serve 1 year in prison (on the lesser charge).**

Prisoner's Dilemma

The rule can be summarized as:

	A cooperates	A defects
B cooperates	A: 1 B: 1	A: 0 B: 3
B defects	A: 3 B: 0	A: 2 B: 2

Cooperate: remain silent
Defect: betray the other

From A's perspective, what should A do?

Prisoner's Dilemma

The rule can be summarized as:

	A cooperates	A defects
B cooperates	A: 1 B: 1	A: 0 B: 3
B defects	A: 3 B: 0	A: 2 B: 2

Cooperate: remain silent
Defect: betray the other

From A's perspective, what should A do?

Prisoner's Dilemma

The rule can be summarized as:

	A cooperates	A defects
B cooperates	A: 1 B: 1	A: 0 B: 3
B defects	A: 3 B: 0	A: 2 B: 2

Cooperate: remain silent
Defect: betray the other

From A's perspective, what should A do?

Prisoner's Dilemma

The rule can be summarized as:

	A cooperates	A defects
B cooperates	A: 1 B: 1	A: 0 B: 3
B defects	A: 3 B: 0	A: 2 B: 2

No matter what B does, A is better off defecting (same goes for B's perspective).

Cooperate: remain silent
Defect: betray the other

Rational agent: an agent that acts in a way to maximize its performance (minimize the sentence time in Prisoner's Dilemma) with all possible action.

Prisoner's Dilemma

The rule can be summarized as:

	A cooperates	A defects
B cooperates	A: 1 B: 1	A: 0 B: 3
B defects	A: 3 B: 0	A: 2 B: 2

Cooperate: remain silent
Defect: betray the other

However, prisoners are not achieving the best outcome they both want – set free, when acting rationally as we analyze.

Prisoner's Dilemma

Game theory tells us what a perfectly rational agent should do. It is much harder to predict what real people actually do. In real experiments, **subjects are found to cooperate much more than the rational agent model predicts.**

Is it because people are not smart enough to understand the game scenario?

Is it because people are knowingly acting contrary to their own interest?

Prisoner's Dilemma

Altruism

Many people are willing to incur a cost to themselves to benefit another person.

But again, why is that?

Altruism seems to be counterproductive if, natural selection, animals are in constant competition with each other to survive and reproduce.

In a population where some people help others, even to their own detriment, and others are purely selfish, it seems like the selfish ones would benefit, the altruistic ones would suffer, and the genes for altruism would be driven to extinction.

Prisoner's Dilemma

THE UNIVERSITY OF WESTERN AUSTRALIA

Problem of Altruism

Why haven't the genes for altruism died out?

Probably, altruism is **adaptive**.

In other words, maybe genes for altruism make people more likely to survive and reproduce.

To test this assumption, we can use the Prisoner's Dilemma and conduct a simulation.

THE UNIVERSITY OF WESTERN AUSTRALIA

Prisoner's Dilemma Tournaments

In the late 1970s Robert Axelrod, a political scientist at the University of Michigan, organized a tournament to compare strategies for playing Prisoner's Dilemma (PD).

He invited participants to submit strategies in the form of computer programs, then played the programs against each other and kept score.

Specifically, they played the iterated version of PD, in which the agents play multiple rounds against the same opponent, so their decisions can be based on history.

Prisoner's Dilemma

Tit for tat (TFT)

TFT always cooperates during the first round of an iterated match; after that, it copies whatever the opponent did during the previous round.

Properties of Good Strategies:

- **Nice**: The strategies that do well cooperate during the first round, and generally cooperate as often as they defect in subsequent rounds.
- **Retaliating**: Strategies that cooperate all the time did not do as well as strategies that retaliate if the opponent defects.
- **Forgiving**: But strategies that were too vindictive tended to punish themselves as well as their opponents.
- **Non-envious**: Some of the most successful strategies seldom outscore their opponents; they are successful because they do well enough against a wide variety of opponents.

Simulating Evolution of Cooperation

THE UNIVERSITY OF WESTERN AUSTRALIA

Questions we may try to answer with simulating PD's Tournaments

- How the distribution of strategies change over time in a population of PD tournaments?
- Can genes for niceness, retribution, and forgiveness appear by mutation?
- Can they successfully invade a population of other strategies?
- Can they resist being invaded by subsequent mutations?

Simulating Evolution of Cooperation

Encode a PD strategy as a genotype

```
1  class Agent:
2
3      keys = [(None, None),
4              (None, 'C'),
5              (None, 'D'),
6              ('C', 'C'),
7              ('C', 'D'),
8              ('D', 'C'),
9              ('D', 'D')]
10
11     def __init__(self, values, fitness=np.nan):
12         """Initialize the agent.
13
14         values: sequence of 'C' and 'D'
15         """
16         self.values = values
17         self.responses = dict(zip(self.keys, values))
18         self.fitness = fitness
```

TFT

C
C
D
C
D
C
D

Map the agent's choice in each round to the opponent's choice in the previous two rounds.

The genotype of a strategy is a sequence of choice with each item corresponds to the opponent's different choice.

Simulating Evolution of Cooperation

Agents can copy with a probability of mutation

```
def copy(self, prob_mutate=0.05):
    """Make a copy of this agent.
    """
    if np.random.random() > prob_mutate:
        values = self.values
    else:
        values = self.mutate()
    return Agent(values, self.fitness)
```

Mutation works by choosing a random value in the genotype and flipping from 'C' to 'D' or vice versa.

```
def mutate(self):
    """Makes a copy of this agent's values, with one mutation.

    returns: sequence of 'C' and 'D'
    """
    values = list(self.values)
    index = np.random.choice(len(values))
    values[index] = 'C' if values[index] == 'D' else 'D'
    return values
```

Simulating Evolution of Cooperation

The Tournament class encapsulates the details of the PD competition:

```
class Tournament:

    payoffs = {('C', 'C'): (3, 3),
               ('C', 'D'): (0, 5),
               ('D', 'C'): (5, 0),
               ('D', 'D'): (1, 1)}

    num_rounds = 6

    def play(self, agent1, agent2):
        """Play a sequence of iterated PD rounds.

        agent1: Agent
        agent2: Agent

        returns: tuple of agent1's score, agent2's score
        """
        agent1.reset()
        agent2.reset()

        for i in range(self.num_rounds):
            resp1 = agent1.respond(agent2)
            resp2 = agent2.respond(agent1)

            pay1, pay2 = self.payoffs[resp1, resp2]

            agent1.append(resp1, pay1)
            agent2.append(resp2, pay2)

        return agent1.score, agent2.score
```

a dictionary that maps from the agents' choices to their rewards.

Initializes the agents before the first round, resetting their scores and the history of their responses.

```
def reset(self):
    """Reset variables before a sequence of games.
    """
    self.hist = [None, None]
    self.score = 0
```

Simulating Evolution of Cooperation

THE UNIVERSITY OF WESTERN AUSTRALIA

The Tournament class encapsulates the details of the PD competition:

```
class Tournament:

    payoffs = {('C', 'C'): (3, 3),
               ('C', 'D'): (0, 5),
               ('D', 'C'): (5, 0),
               ('D', 'D'): (1, 1)}

    num_rounds = 6

    def play(self, agent1, agent2):
        """Play a sequence of iterated PD rounds.

        agent1: Agent
        agent2: Agent

        returns: tuple of agent1's score, agent2's score
        """
        agent1.reset()
        agent2.reset()
        for i in range(self.num_rounds):
            resp1 = agent1.respond(agent2)
            resp2 = agent2.respond(agent1)

            pay1, pay2 = self.payoffs[resp1, resp2]

            agent1.append(resp1, pay1)
            agent2.append(resp2, pay2)

        return agent1.score, agent2.score
```

Asks each agent for their response, given the opponent's previous responses.

```
keys = [(None, None),
        (None, 'C'),
        (None, 'D'),
        ('C', 'C'),
        ('C', 'D'),
        ('D', 'C'),
        ('D', 'D')]
```

```
def past_responses(self, num=2):
    """Select the given number of most recent responses.

    num: integer number of responses

    returns: sequence of 'C' and 'D'
    """
    return tuple(self.hist[-num:])

def respond(self, other):
    """Choose a response based on the opponent's recent responses.

    other: Agent

    returns: 'C' or 'D'
    """
    key = other.past_responses()
    resp = self.responses[key]
    return resp
```

Simulating Evolution of Cooperation

The Tournament class encapsulates the details of the PD competition:

```
class Tournament:

    payoffs = {('C', 'C'): (3, 3),
               ('C', 'D'): (0, 5),
               ('D', 'C'): (5, 0),
               ('D', 'D'): (1, 1)}

    num_rounds = 6

    def play(self, agent1, agent2):
        """Play a sequence of iterated PD rounds.

        agent1: Agent
        agent2: Agent

        returns: tuple of agent1's score, agent2's score
        """
        agent1.reset()
        agent2.reset()

        for i in range(self.num_rounds):
            resp1 = agent1.respond(agent2)
            resp2 = agent2.respond(agent1)

            pay1, pay2 = self.payoffs[resp1, resp2]

            agent1.append(resp1, pay1)
            agent2.append(resp2, pay2)

        return agent1.score, agent2.score
```

Updates each agent by storing the choices and adding up the scores from successive rounds.

```
def append(self, resp, pay):
    """Update based on the last response and payoff.

    resp: 'C' or 'D'
    pay: number
    """
    self.hist.append(resp)
    self.score += pay
```

Simulating Evolution of Cooperation

The Tournament class encapsulates the details of the PD competition:

```
def melee(self, agents, randomize=True):
    """Play each agent against two others.

    Assigns the average score from the two games to agent.fitness

    agents: sequence of Agents
    randomize: boolean, whether to shuffle the agents
    """
    if randomize:
        agents = np.random.permutation(agents)

    n = len(agents)
    i_row = np.arange(n)
    j_row = (i_row + 1) % n

    totals = np.zeros(n)

    for i, j in zip(i_row, j_row):
        agent1, agent2 = agents[i], agents[j]
        score1, score2 = self.play(agent1, agent2)
        totals[i] += score1
        totals[j] += score2

    for i in i_row:
        agents[i].fitness = totals[i] / self.num_rounds / 2
```

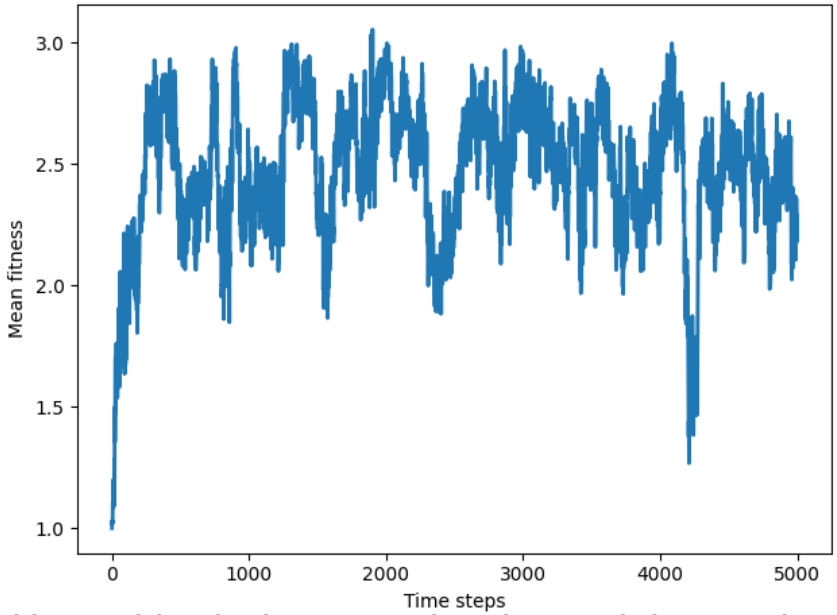
Simulating Evolution of Cooperation

Simulation of PD Tournament

```
1  class PDSimulation(Simulation):
2
3      def __init__(self, tournament, agents):
4          """Create the simulation:
5
6          tournament: Tournament object
7          agents: sequence of agents
8          """
9          self.tournament = tournament
10         self.agents = np.asarray(agents)
11         self.instruments = []
12
13     def step(self):
14         """Simulate a time step and update the instruments.
15         """
16         self.tournament.melee(self.agents)
17         Simulation.step(self)
18
19     def choose_dead(self, fits):
20         """Choose which agents die in the next timestep.
21
22         fits: fitness of each agent
23
24         returns: indices of the chosen ones
25         """
26         ps = prob_survive(fits)
27         n = len(self.agents)
28         is_dead = np.random.random(n) < ps
29         index_dead = np.nonzero(is_dead)[0]
30         return index_dead
```

Simulating Evolution of Cooperation

Quantify the fitness

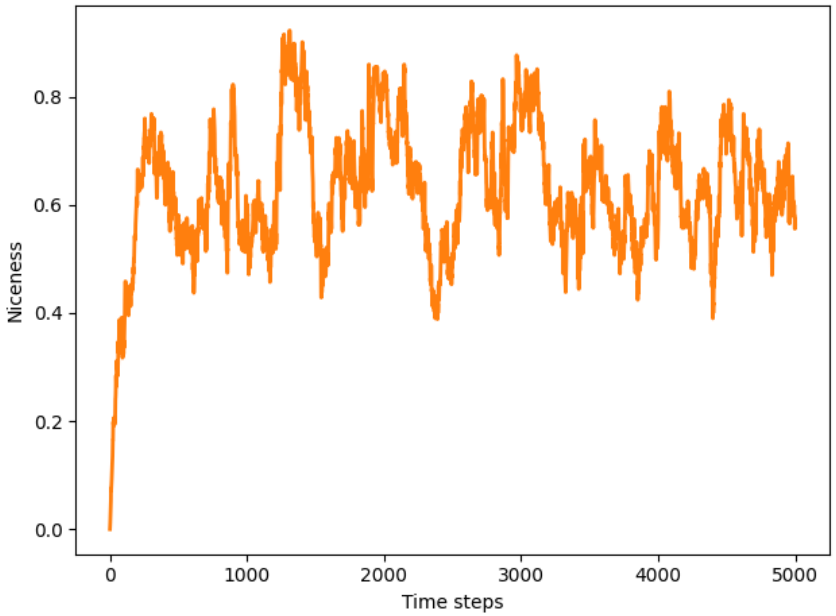


Results of starting with 100 identical agents who always defect, and run the simulation for 5000 steps.

Initially mean fitness is 1, because when defectors face each other, they get only 1 point each per round. After about 500 time steps, mean fitness increases to nearly 3, which is what cooperators get when they face each other.

Simulating Evolution of Cooperation

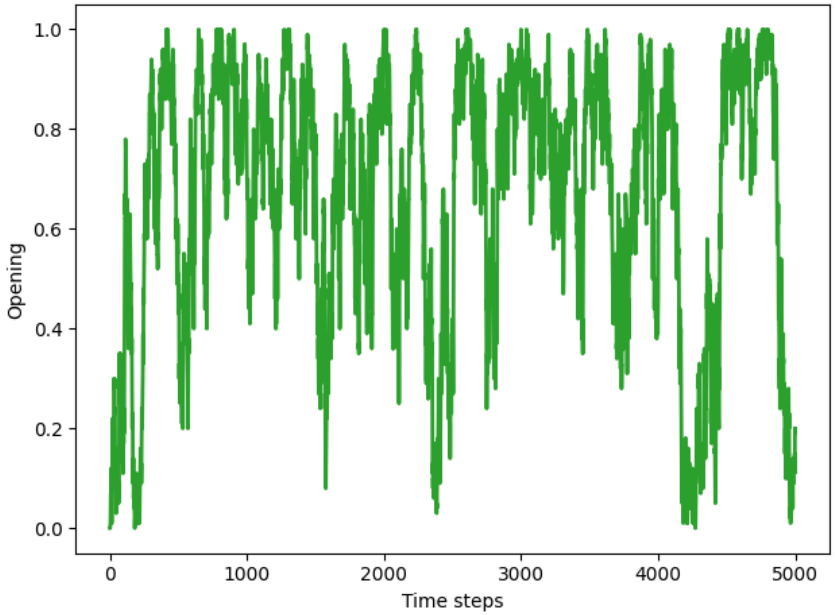
Quantify the Niceness



Niceness measures the fraction of cooperation in the genotypes of the agents after each time step.

Simulating Evolution of Cooperation

Quantify the Opening



Niceness measures the fraction of agents who cooperate in the first round.

Simulating Evolution of Cooperation

Conclusion

The agents in these simulations are simple, and the Prisoner's Dilemma is a highly abstract model of a limited range of social interactions.

- Populations of defectors are vulnerable to invasion by nicer strategies.
- The average amount of niceness is generally high.
- The average level of fitness is generally closer to a utopia of cooperation.
- Some degree of retaliation may be adaptive, but it might not be necessary for all agents to retaliate.
- If there is enough retaliation in the population as a whole, that might be enough to prevent invasion by defectors.

NEXT WEEK PREVIEW…

THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 12: Self-organized Criticality

- 2D Cellular Automata
  - Percolation model
  - Fractal detection
  - Sand pile model
- Final Exam Information