Assignment on Computational Model of Trust and Reputation

Sovan Sahoo

February 2024

Introduction

Consider two agents, denoted as a and b, interacting within a specific context c. This computational model aims to estimate the reputation (θ_{ab}) of agent b in the eyes of agent a. The model focuses on the reciprocal cooperation between the two agents and utilizes a history of encounters to infer trust and reputation.

Trust Definition

Trust (τ) in context c is defined as the expected value of the reputation parameter ($\theta(c)$) given the history of encounters ($D(c)$):

$$\tau(c) = E[\theta(c)|D(c)]$$

Here:

- $\theta(c)$ is the reputation parameter in context c.
- $D(c)$ is the history of encounters within context c.
- $E[\cdot|\cdot]$ denotes the expected value conditional on the provided history of encounters.

This expression represents the anticipated or average level of cooperation from agent b towards agent a within the specific context c, based on historical data.

Computational Model

In the computational model, we assume that agent a consistently performs "cooperate" actions. The encounters between a and b are represented by binary random variables $x_{ab}(i)$, where $x_{ab}(i) = 1$ if b cooperates with a and $x_{ab}(i) = 0$ otherwise. The history of n previous encounters is denoted as $D_{ab} = \{x_{ab}(1), x_{ab}(2), \ldots, x_{ab}(n)\}$.

The reputation parameter θ_{ab} is estimated based on the proportion of cooperative actions over all n encounters. This proportion is modeled using a Beta distribution:

$$p(\theta) = \mathrm{Beta}(c_1, c_2)$$

The estimator for θ_{ab} is then given by:

$$p(\theta|D) = \mathrm{Beta}(c + p, c + n - p)$$

Assuming independence of cooperation probabilities between encounters, the likelihood of p cooperations and $(n–p)$ defections is modeled as:

$$L(D_{ab}|\theta) = \theta^p(1 - \theta)^{n-p}$$

The Beta distribution is chosen as the conjugate prior for this likelihood. Combining the prior and the likelihood, the posterior estimate for θ is obtained.

Modification and Application

Let T_a represent the trust of agent a in the context of interactions with agent b. In a given trial, both agents a and b have the binary choices of cooperation (C) or defection (D). The reputation of agent b, denoted as R_b, is defined as the ratio of agent b's payoff (P_b) to the total payoff (P_{total}) in that specific trial where payoff was calculated from the payoff matrix similar to what used in prisoners dilemma.

$$R_b = \frac{P_a}{P_{\text{total}}}$$

The total payoff in a trial is the sum of the individual payoffs of both agents a and b:

$$P_{\text{total}} = P_a + P_b$$

The trust of agent a (T_a) is then determined as the expectation of the reputation of agent b:

$$T_a = \mathbb{E}[R_b]$$

Trust Calculation Scenarios

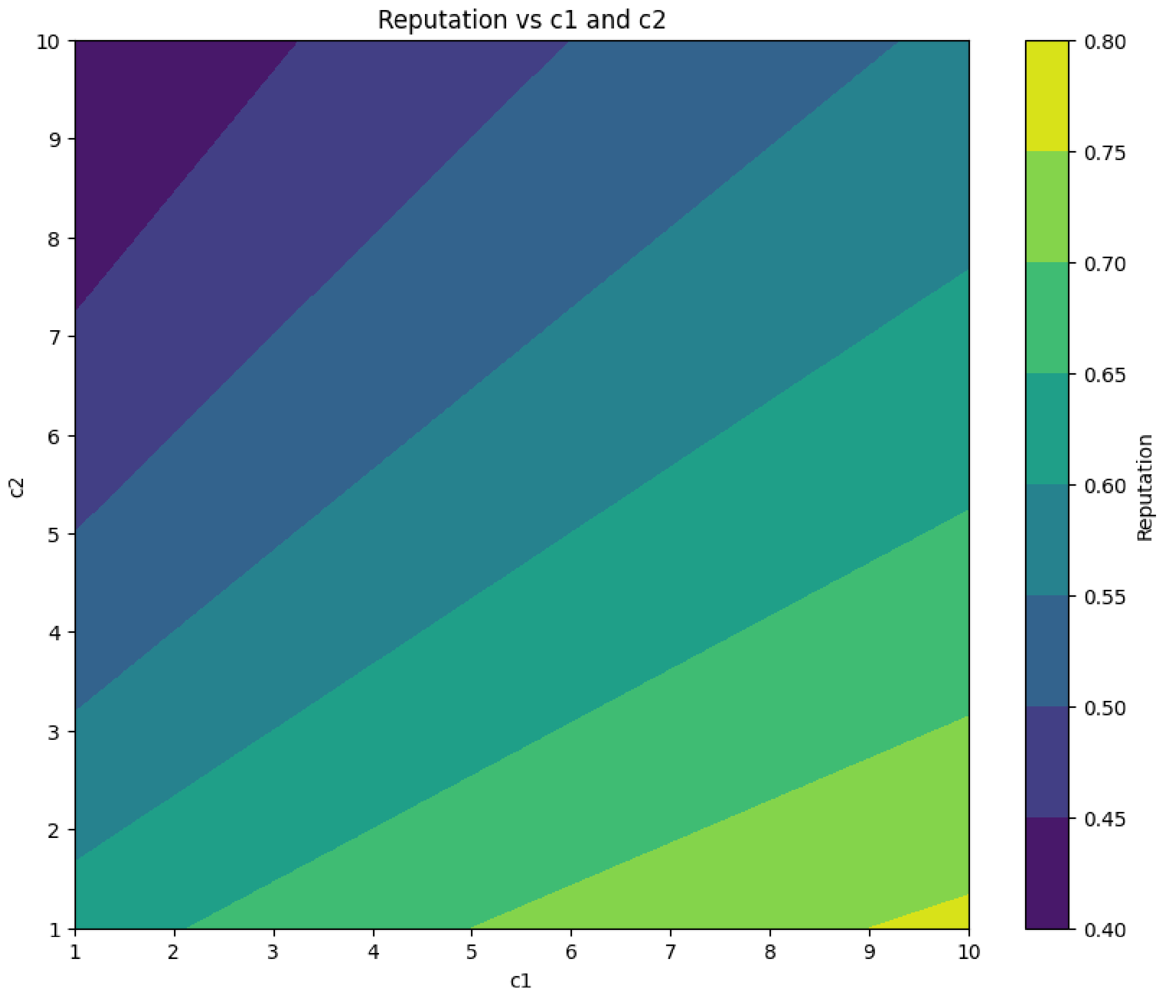
Trust was calculated in four scenarios:

1. **Tit for Tat Strategy:**
2. **Tit for Tat Strategy with a Threshold:**
3. **One Agent Always Forgiving (Co-operating):**
4. **Complete Co-operation:**

Code along with Plot Generated for the original Paper and Modified Application

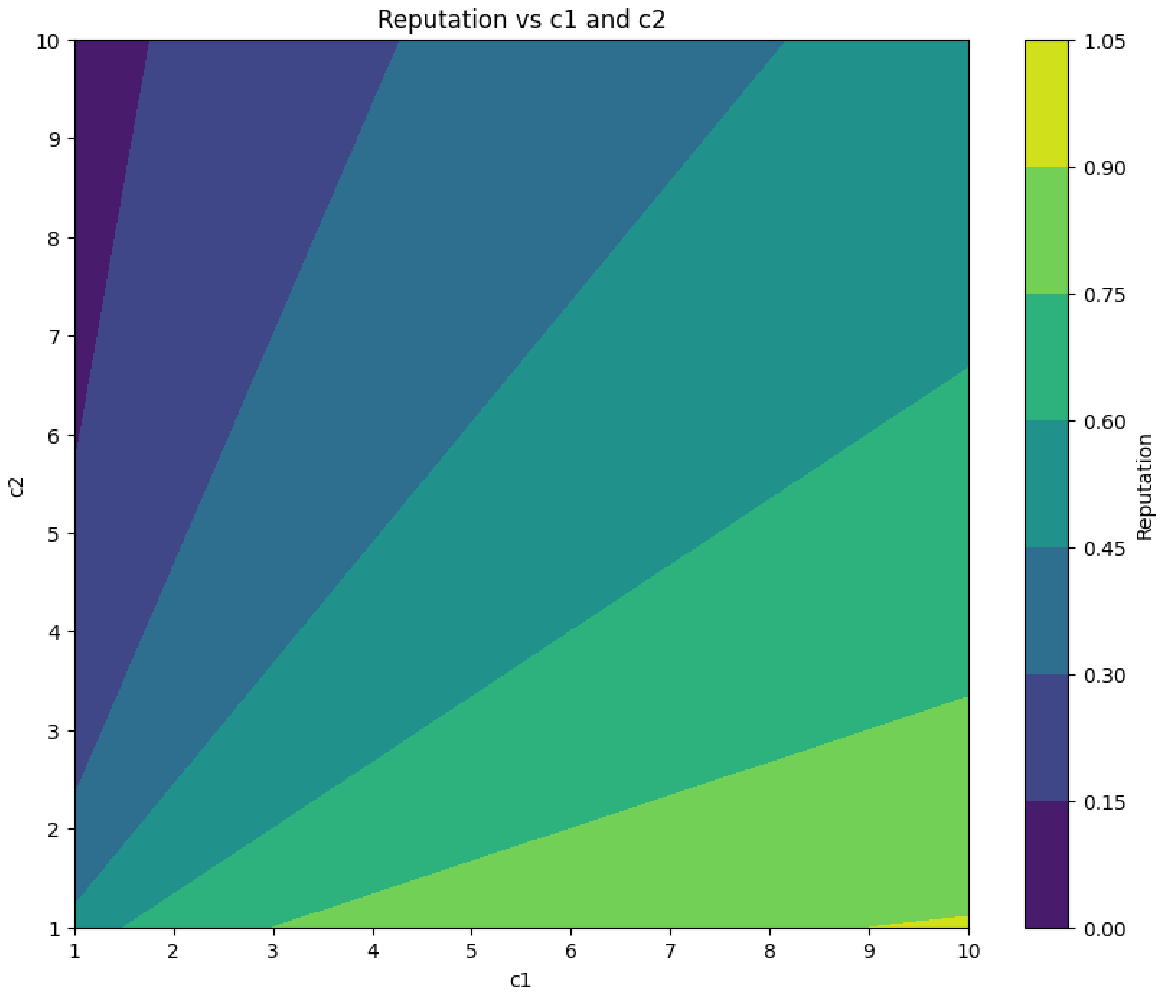
In [1]:

```
import numpy as np
from scipy.stats import beta
import matplotlib.pyplot as plt

def calculate_conditional_expectation(history, c1, c2):
    p = sum(history)
    n = len(history)
    posterior_alpha = c1 + p
    posterior_beta = c2 + n - p
    return beta.mean(posterior_alpha, posterior_beta)


history = [1, 1, 0, 1, 0,1,1,1,1,1,0,0,0,1]

c1_values = np.linspace(1, 10, 50)
c2_values = np.linspace(1, 10, 50)
c1_mesh, c2_mesh = np.meshgrid(c1_values, c2_values)
trust_values = np.empty_like(c1_mesh, dtype=float)
for i in range(c1_mesh.shape[0]):
    for j in range(c1_mesh.shape[1]):
        trust_values[i, j] = calculate_conditional_expectation(history, c


plt.figure(figsize=(10, 8))
contour = plt.contourf(c1_mesh, c2_mesh, trust_values, cmap='viridis')
plt.colorbar(contour, label='Reputation')
plt.xlabel('c1')
plt.ylabel('c2')
plt.title('Reputation vs c1 and c2')
plt.show()
```



In [2]:

```
import numpy as np
import matplotlib.pyplot as plt

def calculate_reputation(c1, c2):
    return c1 / (c1 + c2)

c_values = np.linspace(1, 10, 50)
c1_mesh, c2_mesh = np.meshgrid(c_values, c_values)

reputation_values = calculate_reputation(c1_mesh, c2_mesh)

plt.figure(figsize=(10, 8))
contour = plt.contourf(c1_mesh, c2_mesh, reputation_values, cmap='viridis
plt.colorbar(contour, label='Reputation')
plt.xlabel('c1')
plt.ylabel('c2')
plt.title('Reputation vs c1 and c2')
plt.show()
```


Reputation vs c1 and c2
c2
c1
Reputation
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00

In [ ]:

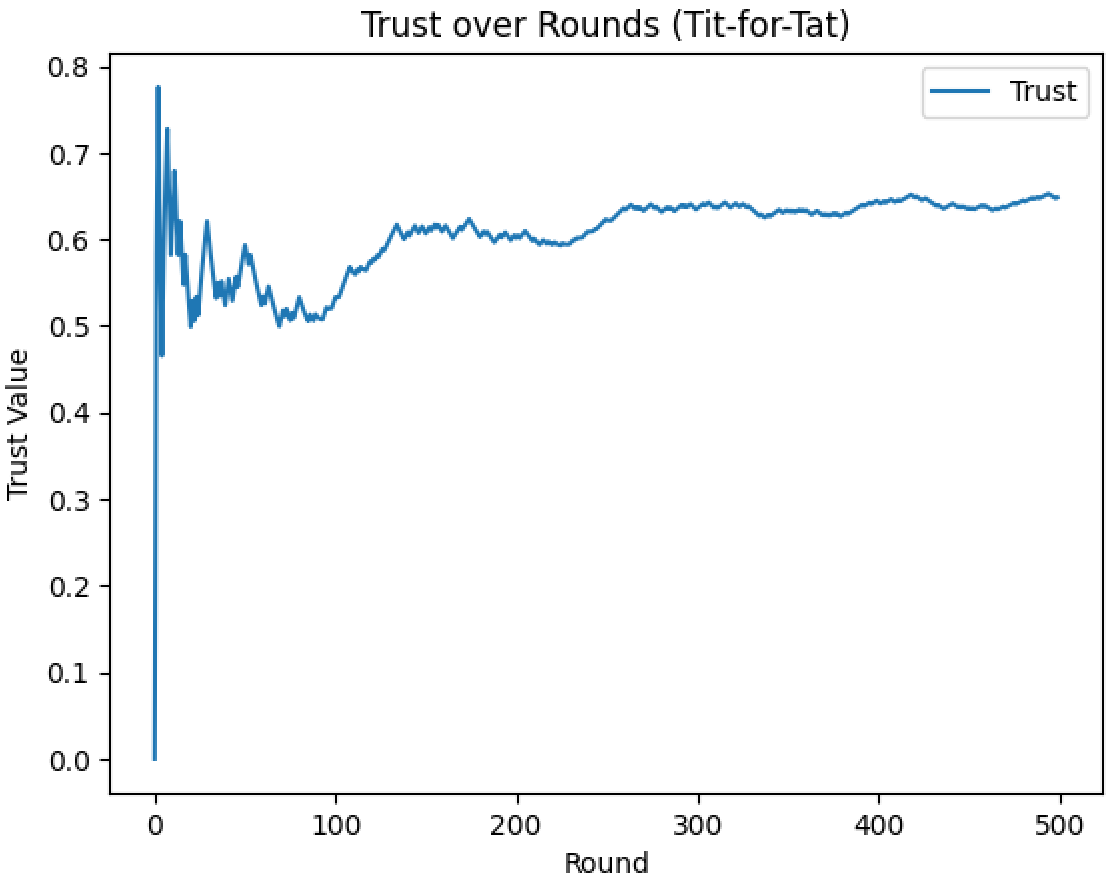
In [1]:
```python
import random
from scipy.stats import norm
import matplotlib.pyplot as plt  # Don't forget to import matplotlib
from scipy.stats import beta
import numpy as np
```

In [2]:
```python
def tit_for_tat(num_trials):

    player_a_moves = ["C"]
    player_b_moves = ["C"]

    for _ in range(num_trials - 1):

        if player_b_moves[-1] == "C":
            player_a_moves.append("C")
        else:
            player_a_moves.append("D")


        opponent_response = random.choice(["C", "D"])
        player_b_moves.append(opponent_response)

    return player_a_moves, player_b_moves

def tit_for_tat_with_threshold(num_rounds, threshold):
    actions_a = []
    actions_b = []

    for round_num in range(1,num_rounds+1):
        if round_num <= threshold:
            actions_a.append("C")
        else:

            if actions_b.count("D") > threshold:
                actions_a.append("D")
            else:
                actions_a.append("C")

        actions_b.append("D")

    return actions_a, actions_b


def one_cooperating_strategy(num_trials):
    player_d_moves=[]
    player_c_moves=[]
    for _ in range(num_trials):

        player_d_moves.append(random.choice(["C", "D"]))
        player_c_moves.append("C")
    return player_d_moves, player_c_moves

def both_cooperating_strategy(num_trials):
    player_e_moves=[]
    player_f_moves=[]
    for _ in range(num_trials):

        player_e_moves.append("C")
```

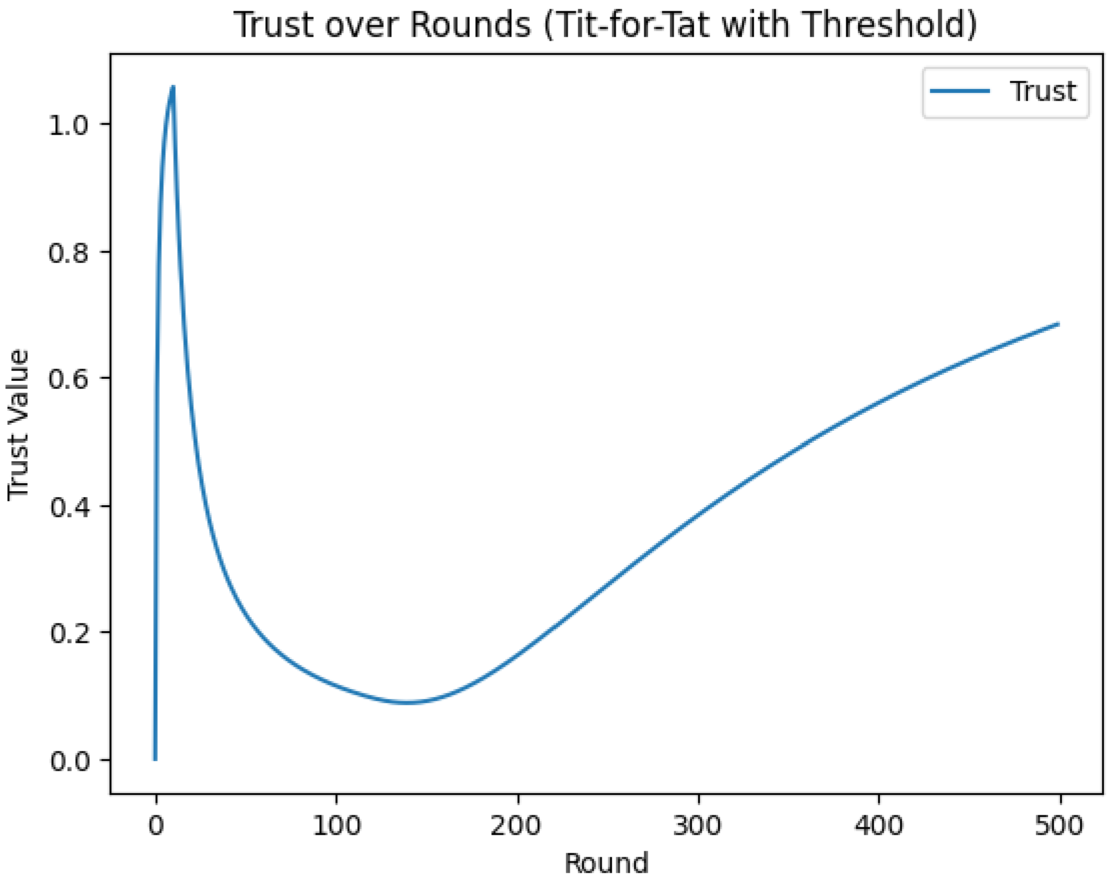
```
        player_f_moves.append("C")
    return player_e_moves, player_f_moves

payoff_matrix = {
    "CC": (1, 1),  # Payoff for mutual cooperation
    "CD": (-10, 10),  # Payoff for A: Cooperation, B: Defection
    "DC": (10, -10),  # Payoff for A: Defection, B: Cooperation
    "DD": (-1, -1)   # Payoff for mutual defection
}
# Simulate 500 trials
num_trials = 500
player_a, player_b = tit_for_tat(num_trials)
player_a1, player_b1 = tit_for_tat_with_threshold(num_trials, threshold=9
player_a2, player_b2 = one_cooperating_strategy(num_trials)
player_a3, player_b3 = both_cooperating_strategy(num_trials)

mu3 = 4
mu4 =0
sigma = 1
pdf_values=[]
pdf_values1=[]
pdf_values2=[]
pdf_values3=[]
```

In [3]:

```
# For the tit-for-tat
payoffs_a = []
payoffs_b = []
trust_values = []
rep_values=[]
for i in range(len(player_a)):
    action_a = player_a[i]
    action_b = player_b[i]
    key = action_a + action_b
    payoffs_a.append(payoff_matrix[key][0])
    payoffs_b.append(payoff_matrix[key][1])

    if (sum(payoffs_a)+sum(payoffs_b)) == 0:
        rep_values.append(0.5)  # Set trust value to a default value
    else:
        rep_values.append(sum(payoffs_a) / (sum(payoffs_a)+sum(payoffs_b)

for i in range(len(player_a)):

    pdf_value = beta.pdf(rep_values[i], 4.5, 4.5)
    pdf_values.append(pdf_value)

for i in range(len(rep_values)):
    partial_expectation = sum(pdf * rep_value for pdf, rep_value in zip(p
    trust_values.append(partial_expectation)

plt.plot(range(len(trust_values)), trust_values, label='Trust')
plt.xlabel('Round')
plt.ylabel('Trust Value')
plt.title('Trust over Rounds (Tit-for-Tat)')
plt.legend()
plt.show()
```



In [4]:
```
# For the tit-for-tat with threshold strategy
payoffs_a1 = []
payoffs_b1 = []
trust_values1 = []
rep_values1=[]
for i in range(len(player_a1)):
    action_a1 = player_a1[i]
    action_b1 = player_b1[i]
    key1 = action_a1 + action_b1
    payoffs_a1.append(payoff_matrix[key1][0])
    payoffs_b1.append(payoff_matrix[key1][1])

    if (sum(payoffs_a1)+sum(payoffs_b1)) == 0:
        rep_values1.append(0.5)  # Set trust value to a default value
    else:
        rep_values1.append(sum(payoffs_a1) / (sum(payoffs_a1)+sum(payoffs
for i in range(len(player_a)):

    pdf_value1 = beta.pdf(rep_values1[i], 4.5, 4.5)
    pdf_values1.append(pdf_value1)
for i in range(len(rep_values1)):
    partial_expectation1 = sum(pdf * rep_value for pdf, rep_value in zip(
    trust_values1.append(partial_expectation1)


# Plot Trust values for tit-for-tat with threshold
plt.plot(range(len(trust_values1)), trust_values1, label='Trust')
plt.xlabel('Round')
plt.ylabel('Trust Value')
plt.title('Trust over Rounds (Tit-for-Tat with Threshold)')
plt.legend()
plt.show()
```

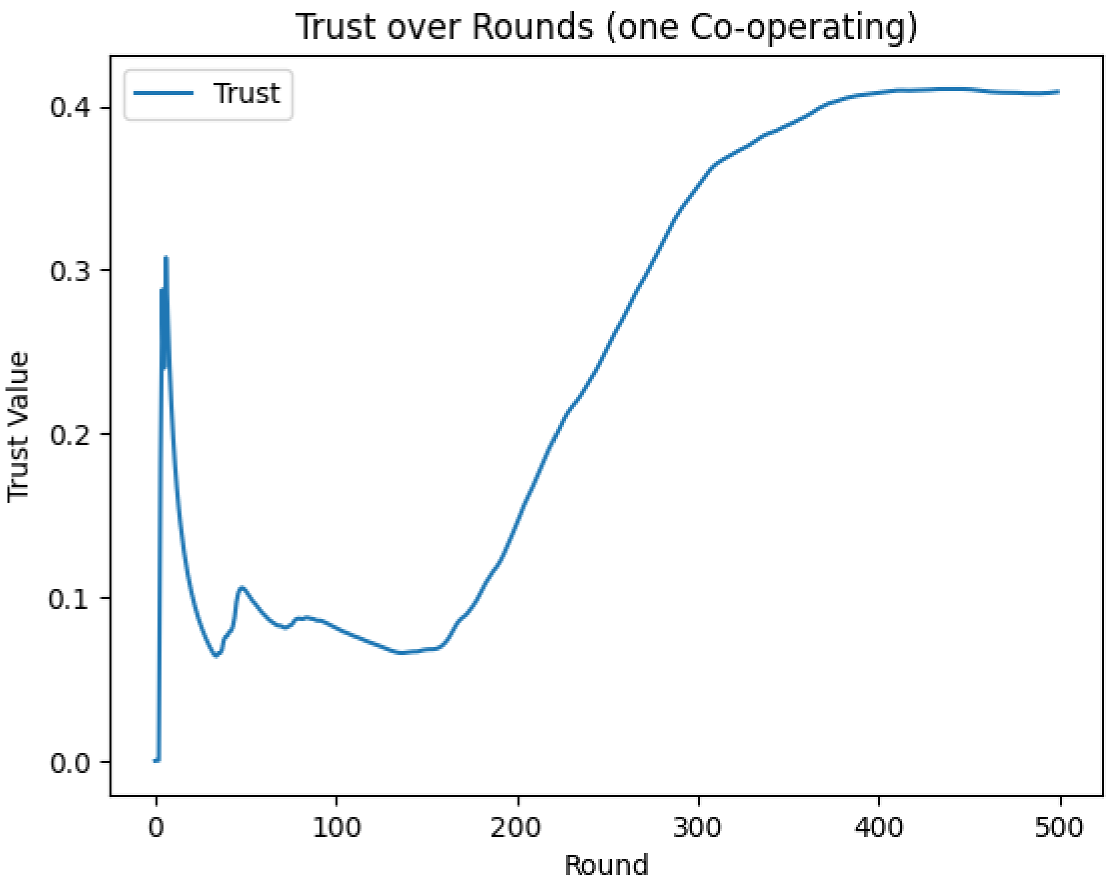


In [5]:

```
# For ever co-operating
payoffs_a2= []
payoffs_b2= []
trust_values2 = []
rep_values2=[]
for i in range(len(player_a2)):
    action_a2 = player_a2[i]
    action_b2 = player_b2[i]
    key = action_a2 + action_b2
    payoffs_a2.append(payoff_matrix[key][0])
    payoffs_b2.append(payoff_matrix[key][1])

    if (sum(payoffs_a2)+sum(payoffs_b2)) == 0:
        rep_values2.append(0.5)  # Set trust value to a default value or
    else:
        rep_values2.append(sum(payoffs_a2) / (sum(payoffs_a2)+sum(payoffs

print(np.median(rep_values2))
for i in range(len(player_a2)):
    # Calculate the PDF values using the normal distribution
    pdf_value2 = norm.pdf(rep_values2[i], mu3, sigma)
    pdf_values2.append(pdf_value2)

for i in range(len(rep_values2)):
    partial_expectation2 = sum(pdf * rep_value for pdf, rep_value in zip(
    trust_values2.append(partial_expectation2)

# Plot Trust values for tit-for-tat with threshold
plt.plot(range(len(trust_values2)), trust_values2, label='Trust')
plt.xlabel('Round')
plt.ylabel('Trust Value')
plt.title('Trust over Rounds (one Co-operating)')
```

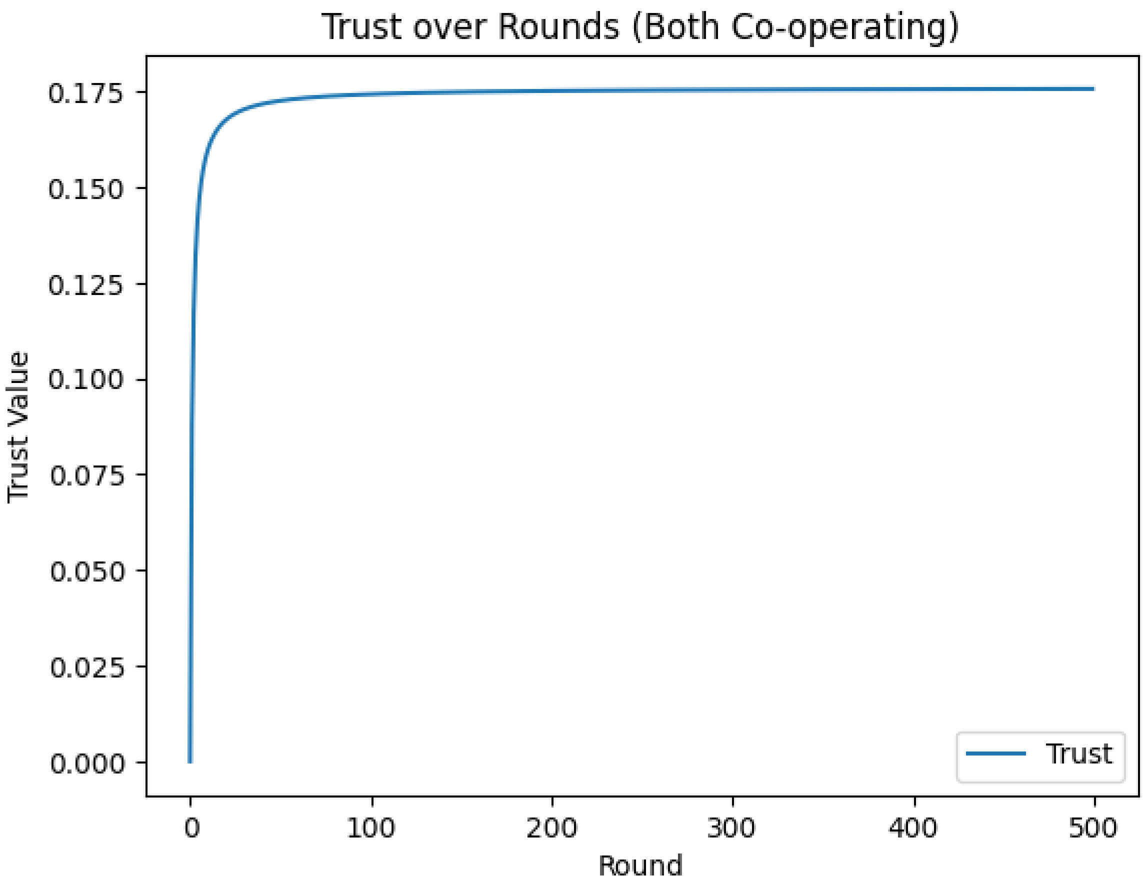
```
plt.legend()
plt.show()
```

5.830851396540525



In [7]:
```
# For ever co-operating
payoffs_a3= []
payoffs_b3= []
trust_values3 = []
rep_values3=[]
for i in range(len(player_a3)):
    action_a3 = player_a3[i]
    action_b3 = player_b3[i]
    key = action_a3 + action_b3
    payoffs_a3.append(payoff_matrix[key][0])
    payoffs_b3.append(payoff_matrix[key][1])

    if (sum(payoffs_a3)+sum(payoffs_b3)) == 0:
        rep_values3.append(0.5)  # Set trust value to a default value or
    else:
        rep_values3.append(sum(payoffs_a3) / (sum(payoffs_a3)+sum(payoffs

for i in range(len(player_a3)):
    # Calculate the PDF values using the normal distribution
    pdf_value3 = norm.pdf(rep_values3[i],mu4, sigma)
    pdf_values3.append(pdf_value3)

for i in range(len(rep_values3)):
    partial_expectation3 = sum(pdf * rep_value for pdf, rep_value in zip(
    trust_values3.append(partial_expectation3)

# Plot Trust values for tit-for-tat with threshold
plt.plot(range(len(trust_values3)), trust_values3, label='Trust')
```

```python
plt.xlabel('Round')
plt.ylabel('Trust Value')
plt.title('Trust over Rounds (Both Co-operating)')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]: