# Social Incentive Payoff Mechanism

**PART 1. SYSTEM**

1. Each participant *i* has a strategy *xi* - amount of workload the participant wants to perform
2. Further, define X as all participants' strategies (i.e., strategy profiles) and X–i as the strategies excluding participant i. As the quality of sensing tasks are interdependent, the utility that participant i can receive is dependent on the strategies and cost of other players.
3. *cost(xi)* → Cost of participating in the game
4. Social Welfare → Sum of all (participants payoffs - cost of participation)

**PART 2. SOCIAL INCENTIVE MECHANISM**

1. Each person i is connected to some random number of other players, who they can influence during the game
2. Each participant can apply some "pressure" to other players to make them conform to the most optimal strategy
3. Use backward induction to find out the strategy which maximizes the "social welfare"

1. Model the mobile crowdsensing problem as three stage stackelberg game.
    a. Platform is the leader, gives various tasks
    b. Participants are the followers
    c. Platform declares an incentive mechanism
2. Participants determine how much pressure to put on their friends (this is the Social Incentive Mechanism)

**LIMITATIONS**

**QUESTIONS TO ASK**

1. What new twist can we add to this problem?
2. Simulations with data?
3. Some other kind of incentive instead of social incentive
4. auction
5. Define the set of participants, N, and each participant's strategy, xi, as well as the cost and utility functions for each participant.
6. Define the social network connecting the participants, where each participant has links to their social friends.
7. Implement a mechanism for updating each participant's strategy based on their own utility as well as the utilities of their social friends.
8. Run the simulation for multiple iterations to observe the evolution of strategies and social welfare over time.
9. Analyze the results to determine the effectiveness of the social incentive mechanism in promoting cooperation and increasing social welfare.

Note that the specific implementation details may vary depending on the specific scenario and assumptions made.

```
import numpy as np

# Define number of participants and sensing reports
n = 5  # number of participants
m = 10  # number of sensing reports

# Define cost and utility functions
def cost(x):
    # Convex cost function
    return x ** 2

def utility(x, X):
    # Concave utility function
    return np.prod(X) ** 0.5 * x ** 0.5

# Define social network as an adjacency matrix
social_network = np.array([
    [0, 1, 1, 0, 0],
    [1, 0, 1, 1, 1],
    [1, 1, 0, 1, 0],
    [0, 1, 1, 0, 1],
    [0, 1, 0, 1, 0]
])

# Define initial strategies for each participant
strategies = np.zeros(n)

# Define function to compute payoffs for each participant
def compute_payoffs(strategies):
    payoffs = np.zeros(n)
    for i in range(n):
        X_i = np.delete(strategies, i)
        payoffs[i] = utility(strategies[i], X_i) - cost(strategies[i])
    return payoffs

# Define function to update strategies using the social incentive mechanism
def update_strategies(strategies, social_network, alpha=0.1):
    new_strategies = np.zeros(n)
    for i in range(n):
        X_i = np.delete(strategies, i)
        # Compute payoffs for participant i and their social friends
        payoffs_i = compute_payoffs(np.insert(X_i, i, strategies[i]))
```

```
        payoffs_j = np.zeros(n)
        for j in range(n):
            if social_network[i, j] == 1:
                X_j = np.delete(strategies, j)
                payoffs_j[j] = compute_payoffs(np.insert(X_j, j,
strategies[j]))[j]
        # Compute social incentive term
        social_incentive = alpha * np.sum(social_network[i] * (payoffs_j
- payoffs_i))
        # Update participant i's strategy
        new_strategies[i] = strategies[i] + social_incentive
    return new_strategies

# Simulate social incentive mechanism for multiple iterations
for i in range(m):
    strategies = update_strategies(strategies, social_network)
    print(f"Iteration {i+1}: Strategies = {strategies}, Payoffs =
{compute_payoffs(strategies)}")
```