

Weighted Graph-Based Modelling of Relationship Dynamics to Determine Optimal Strategies for Long-Term Payoffs in the Iterated Prisoner's Dilemma

Daniel Pedrosa Wu - 13523099¹

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

danielpedrosawu5705@gmail.com, 13523099@std.stei.itb.ac.id

Abstract—This paper explores the implementation of weighted graph in modelling relationship dynamics in the context of Iterated Prisoner's Dilemma and its impact in long-term success. By integrating relationship dynamics with graph theory, this study shows how the ability to build and sever relationship can majorly influence the evolution of strategies over time. The research investigates the dynamics between cooperative and defective strategy. Result shows that defective strategy offers greater immediate result but falls off quickly while cooperative strategy takes time to build up but offers a more sustainable outcome. It suggests that successful strategies in IPD requires a balance between the two concepts.

Keywords—Cooperation, Defection, Moran Process, Prisoner's Dilemma,

I. INTRODUCTION

In this interconnected world, interactions are rarely isolated events. They hinge on trust, reciprocity, potential for mutual benefits and anticipation of future encounters. Although both parties stand to gain from mutual cooperation, the temptation to further one's own self interest and fear of exploitation are ever-present. Consequently, decision-making are rarely straightforward. Individuals, organization and even nations are constantly faced with decisions that have to balance short-term gains with long-term consequences. These dynamics lie at the heart of many real-world decision-making processes.

At their core, these decisions revolve around the choice between cooperation and defection. Cooperation offers mutual benefits, while defection can yield greater immediate rewards. The field of *game theory* provides a mathematical framework for analyzing interdependent decisions and strategic decision-making. Among its many models, the Prisoner's Dilemma (PD) stands as a fundamental paradigm in understanding the tension between self interest and collective welfare.

In the PD, two players—a term used in game theory to refer to the parties involved—must decide between cooperating with one another for mutual benefit or defecting for personal gain, with the outcome depending on their combined choices. Despite its simplicity, the PD stands out for its relevance in several fields. Even nowadays, it is still used to model real-world scenarios, ranging from economics to international relations and

scientific fields. While the classic version of PD offers valuable insights into one-time interactions, very few real-world interactions are one-off occurrences. Most decisions are influenced by past actions and ongoing relationships.

To better reflect these real scenarios, the PD is often extended into the Iterated Prisoner's Dilemma (IPD). The IPD extends the PD into repeated interactions, allowing players to adapt their strategies and allowing past outcomes to influence future choices. However, traditional IPD models often neglect to account for the possibility of severing and rebuilding relationships. These models assume that players will continue to interact with each other despite a history of negative interactions, without considering that relationship may be severed or rebuilt over time.

In many real-world context, severing ties can be a valid strategy to help mitigate risk and protect personal interest. These scenarios can be modelled using weighted graphs by representing the players as nodes and the strength of the relationship as edges, updating based on cooperative or defective actions. The weights on the edge can reflect the level of trust, cooperation and history between the players. This approach provides a more dynamic system, reflecting the nuances of real-world decision-making processes.

This study explores the application of weighted graphs to enhance the modelling of the Iterated Prisoner's Dilemma. By incorporating the evolving relationships between players into a graph structure, this study seeks to identify strategies that maximizes long-term payoff by balancing short-term decisions with long-term relationship outcomes. Through Python-based simulations, this study aims to simulate the intricate and nuanced dynamics of real-world decision-making.

II. THEORETICAL FOUNDATION

A. Graph

A *graph* is a discrete mathematical structure, consisting of vertices (nodes) and edges that connect these vertices and is used to model relationship between objects [4]. Formally, a graph G is defined as $G = (V, E)$, where:

1. V is a nonempty set of vertices, which represents the objects in the system [1].

$$V = \{v_1, v_2, \dots, v_n\}$$

2. E is a set of edges, which represents the relationship between vertices [1].

$$E = \{e_1, e_2, \dots, e_n\}$$

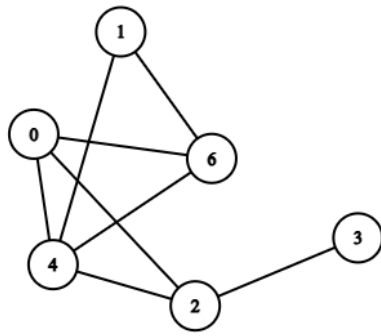


Fig. 2.1. Example of a Graph

Source: Generated using

https://csacademy.com/app/graph_editor/

Graphs can be grouped based on different properties. These properties include:

1. Complexity

A graph in which each pair of vertices are connected by at most one edge is called a *simple graph* [1]. More complex graphs may include loops, where edges connect a vertex to itself or multiple edges, where the same pair of vertices are connected by two or more edges. These graphs are called *unsimple graph* [1], which can further be classified into *multi-graph* that contains multiple edges and *pseudo-graph* which contains loops [1].

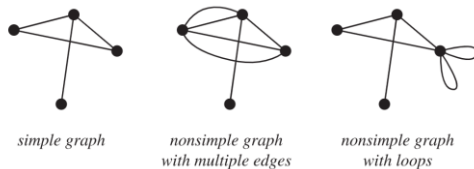


Fig. 2.2. Simple and Unsimple Graphs

Source: <https://mathworld.wolfram.com/SimpleGraph.html>

2. Directional Orientation

A graph in which each edge has a specified direction is called a *directed graph* [1]. This indicates a one-way relation between two vertices. When the edges of a graph do not have a specified direction, the graph is called an *undirected graph* [1]. While they are most commonly used to represent a bidirectional relationship, this is not always the case.

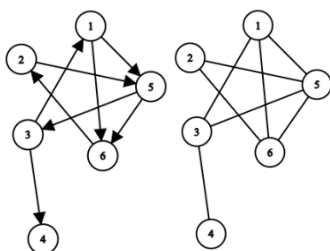


Fig. 2.3. Directed and Undirected Graph

Source: Generated using

https://csacademy.com/app/graph_editor/

3. Edge Weights

A graph in which each edge is assigned a numerical value (weight) is called a *weighted graph* [1]. These values can represent various attributes, depending on context. In contrast, a graph in which edges are not assigned numerical value is called an *unweighted graph*. This graph treat all edges as equal.

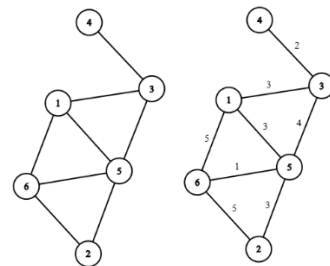


Fig. 2.4. Unweighted and Weighted Graph

Source: Generated using

https://csacademy.com/app/graph_editor/

4. Connectivity

A graph where there exist a path from every vertices to every other vertices is called a *connected graph* [1]. This means that it is possible to reach any vertex to any other vertex by traversing the graph. If there is atleast one pair of vertices that does not satisfy this condition, then the graph is called a *disconnected graph* [1].

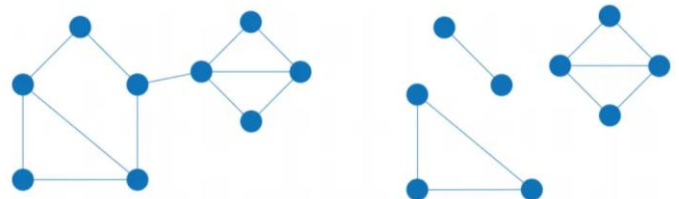


Fig. 2.5. Connected and Disconnected Graph

Source: <https://rajshah001.medium.com/graphs-and-real-life-application-28759b77b833>

Some terminology used in graph theory includes:

1. Adjacency
Two vertices u and v in graph G are called *adjacent* if there exist an edge e of G in which u and v are the endpoints of e [4].
2. Incidency
An edge e is called *incident* with vertices u and v if e connects u and v [4].
3. Degree
The *degree* of a vertex v is the number of edges incident to it, except that a loop at the vertex contributes twice to the degree of the vertex [4].

4. Isolated Vertex

A vertex v is called an *isolated vertex* if there exist no edges e that are incident to it [1].

5. Null Graph

A graph g is called a *null graph* if the set of edges E are empty. In other words, there are no edges e connecting any pair of vertices v in g [1].

6. Path

A *path* in graph G is a sequence of vertices V and edges E such that each pair of vertices are connected by an edge. The path starts at vertex v_0 and ends at vertex v_n [1].

7. Cycle

A *cycle* in graph G is a path that start and end on the same vertex [1].

B. The Prisoner's Dilemma

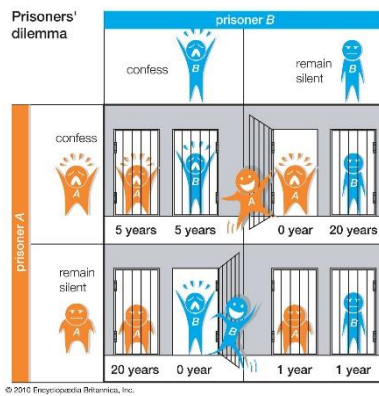


Fig. 2.6. Prisoner's Dilemma Illustrated

Source: <https://www.britannica.com/science/game-theory/The-prisoners-dilemma>

Imagine two individuals, John and Jane who have been arrested for a crime that they committed together. Both of them are placed in separate cells. The police do not have sufficient evidence to convict them of a major crime but they do have sufficient evidence to convict them of a lesser crime. They offer the duo the same deal:

1. If one of them remains silent while the other is silent, then the one who confessed will remain free while the other will receive a long sentence.
2. If both of them confess, then both will receive moderate sentences.
3. If both of them remain silent, then both will receive short sentences.

Both prisoner must decide to cooperate with each other and remain silent or betray the other by confessing. The dilemma that they are faced with is that regardless of what the other chooses, it is within their best interest to betray the other and confess, since the potential reward for confessing is much higher than remaining silent. The rationale is that if Jane cooperates, then John should defect since going free is better than receiving short sentence. If John defects, then Jane should also defect since receiving a short sentence is better than receiving a long sentence. However, notice that if both chooses to betray each

other, they end up with a worse outcome than if both chooses to remain silent.

This puzzle illustrates a conflict between individual rationality and collective well-being. From a self-interested rational perspective, mutual cooperation is irrational. However, if both parties acted on their self-interested rationale, they end up with a worse outcome than if they had both chosen the "irrational" choice of cooperation.

This paradox is also known as the Prisoner's Dilemma (PD), one of the most well-known problem in the field of game theory. In game theory, *rationality* is defined as the ability to make decisions that maximizes one's own personal payoff, based on the given premises and information. In the context of the Prisoner's Dilemma, both players are assumed to be rational agent and are aware that the other is also a rational agent. Given this information, both players are likely choosing to defect since it provides a better payoff regardless of the other player's choice.

In its simplest form, the Prisoner's Dilemma can be expressed using a payoff matrix [5]:

	C	D
C	(R, R)	(S, T)
D	(T, S)	(P, P)

Where:

1. R is the *reward* for mutual cooperation.
2. P is the *punishment* for mutual defection.
3. S is the *sucker* payoff when one player cooperate while the other defect.
4. T is the *temptation* payoff when one player defect while the other cooperate.

This dilemma only works if the payoff follows the inequality $T > R > S > P$. This inequality means that the temptation payoff yields greater reward than the reward for mutual cooperation, while on the other hand the punishment for mutual defection is worse than the sucker payoff. Consequently, the dominant strategy for both player is to defect (D) as it provides the best payoff regardless of the other player's choice. However, in this scenario, both players playing rationally leads to suboptimal outcome for both players, which illustrate the paradoxical nature of this dilemma.

C. The Iterated Prisoner's Dilemma

While the Prisoner's Dilemma involves a one-time interaction, real-world scenarios often involve repeated interactions between individuals. To more accurately model these situations, the Iterated Prisoner's Dilemma (IPD) builds on the classic Prisoner's Dilemma by allowing players to engage in multiple rounds of interactions. This introduces a whole new dimension to the game by allowing players to learn from previous interactions and adapt their strategy accordingly. This version of the Prisoner's Dilemma captures the dynamics of

real-world relationships more accurately.

In contrast to the Prisoner's Dilemma where defecting is the dominant strategy, the IPD allows for more cooperative tactics to emerge as viable and advantageous strategies. The repeated nature of the interactions allow players to influence other's behavior, develop trust and build relationships. In the IPD, players must not only make decisions for immediate payoffs, but also consider the ramifications in future rounds. This creates an environment where past cooperations are rewarded and continuous defections are punished.

It is important to note that this does not mean that defection is an inherently bad strategy. Defection can still be effective in specific context, whether as a means of retaliation or exploit overly trusting players. The iterative nature of the IPD incentivizes player to take a more conservative approach, by weighing the potential risk and reward for each decision. Being overly cooperative can lead to exploitation, while continuous defection leads diminishing trust. Successful players maintain a balance between cooperation and defection, adjusting their strategy based on past actions. In an infinite game, the ability to build and maintain relationship becomes even more important, as opposed to a finite game where player might defect earlier to maximize payoffs. This balancing act emphasizes the complexity of real-world decision-making, where strategies must adapt to evolving circumstances.

In his 1984 book titled *The Evolution of Cooperation*, Robert Axelrod delved into how cooperation can be fostered by self-interested rational individuals. Axelrod detailed how he ran a series of tournaments where individuals submit their strategies for the IPD and pitted against one another in a round robin tournament. Axelrod found that greedy strategies tend to do very poorly in the long run, while more altruistic strategies thrive. To his surprise, the winner was the simplest program, utilizing the *tit for tat* strategy [6]. The strategy starts by cooperating and proceeds to copy the opponent's previous move till the end of the match.

This success can be attributed to the principle of *reciprocity*. Reciprocity refers to responding to another's action with a similar action. By meeting cooperation with cooperation and defection with defection, the strategy fosters trust with other cooperators and deters exploitation by defectors. Based on the tournament results, Axelrod outlined four key elements of a successful strategy [6] which includes:

1. Nice
A successful strategy should always start by cooperating and avoid being the first to defect. This builds trust and encourages mutual cooperation from the start. A successful strategy should also be forgiving, allowing reconciliation but firm enough to avoid being exploited.
2. Non-envious
A successful strategy does not aim to outperform others. Instead, it prioritizes mutually beneficial outcomes, realizing that those benefits accumulate over time.
3. Reciprocating
A successful strategy should reciprocate the opponent's action in kind. It rewards cooperation with cooperation and retaliates against defection to discourage further exploitation.

4. Simple

A successful strategy should be simple and easy to understand. Clearer intentions make it easy for others to cooperate and achieve mutually beneficial outcomes.

D. The Moran Process

The Moran Process, named after British mathematician is a stochastic (random) model used to study evolution within a finite population. Due to being a stochastic model, the outcome of the Moran Process is not deterministic and the system evolves over time in unpredictable ways. The Iterated Prisoner's Dilemma, which focuses on strategy evolution over a series of repeated interactions between two individuals can be further extended to include the Moran Process. By including the stochastic element of the Moran Process, the IPD can be used to not only explore how strategies evolve through repeated interactions, but also how it spreads across a population.

In the context of IPDs, a fixed population of individuals individually adopt their own strategies and interact with one another. Individuals considered *fit* for reproduction are chosen at random based on their payoff from their interactions with other individuals. This individual then *reproduces*, passing down their strategy to another individual chosen at random. Over time, this ensures successful strategies are more likely to spread, while strategies that perform poorly will fade away. In a way, this process reflects real-world scenarios where successful traits, strategies and trends tend to dominate and persist within a population.

III. IMPLEMENTATION

A. Research Limitation

For the purpose of this research, some limitations are set to maintain focus and clarity. The limitations are as follows:

1. Fixed Population Size: This research assumes a fixed population size. The size of the population will not grow or shrink, but the strategies adopted by individuals may change.
2. Finite Iterations: The model is limited to a finite number of iterations due to practical limitations.
3. Simplified Payoff Matrix: A standard payoff matrix following the $T > R > S > P$ inequality is used to simplify interaction dynamics.
4. Limited Strategies: The set of strategies used will be limited to a predefined set, which incorporate cooperative, defective and adaptive strategies to maintain a manageable level of complexity.
5. Limited Interaction Scope: Interactions are restricted to a pair of players at a time, without considering the influence of neighboring individuals. The population only affects the model during reproduction.

B. Programming Language

The implementation of this program will be done using Python. Python is chosen for its versatility, ease of use and extensive library support. A key library used for this implementation is Axelrod. Axelrod is powerful library which provides various tools for Iterated Prisoner's Dilemma research. Axelrod offers over 200 strategies, facilitates the creation of

matches and tournaments, and provides an easy way to analyze the result in detail. Additionally, libraries such as random for its stochastic elements, NetworkX for its graph-based computation and Matplotlib for its visualization will be used to complement Axelrod already robust simulation of the Iterated Prisoner's Dilemma.

```
import axelrod as axl
import random
import networkx as nx
import matplotlib.pyplot as plt
```

C. Simulation Parameters

To effectively model this simulation, a set of parameters is defined. The following parameters control the behavior of the simulation and allows for a more-controlled exploration of relationship dynamics within the scope of the Iterated Prisoner's Dilemma:

1. **THRESHOLD:** A threshold value that determine whether an interaction between two players should occur. If the relationship (edge weights) between two players is below the threshold, relationship will be severed and both will cease to interact. The corresponding edge on the relationship graph will also be removed. This value is used to model a deteriorating relationship.
2. **REBUILD_CHANCE:** This parameter defines the chance of rebuilding a relationship. If a relationship between two players was severed, there is a chance that the relationship might be rebuilt based on this value. This offers a possibility of reconciliation between players.
3. **NUM_ROUNDS:** This parameter dictates the total number of rounds that will take place between the players.
4. **NUM_TURNS:** This parameter specifies the amount of turns in a single round of interaction between each pair of players. In other words, this parameter defines how many moves each player will make in a single round.
5. **USE_MORAN_PROCESS:** This parameter determines if the Moran Process will be used in the simulation. If enabled, players with higher payoffs have a higher chance of reproducing and replacing less fortunate players. This is applied after each rounds.
6. **RANDOM_PLAYERS:** This parameter determines if players are assigned strategies randomly or initialized with a fixed set of strategies.
7. **PLAYER_COUNTS:** This parameter determines the amount of players generated when **RANDOM_PLAYERS** is enabled.
8. **INIT_WEIGHT:** This parameter sets the initial weights of relationship to determine the strength of the relationship. These weights will evolve as the simulation progress.

```
# Simulation parameters
THRESHOLD = 0 # Only allow matches if the weight is above this
threshold
REBUILD_CHANCE = 0.1 # Chance to rebuild severed relationships
NUM_ROUNDS = 5 # Number of rounds in the simulation
NUM_TURNS = 20 # Number of turns each round between each player
USE_MORAN_PROCESS = False # Toggle for Moran process
RANDOM_PLAYERS = False # Toggle for randomized player
PLAYER_COUNT = 10 # Player count for randomize player
```

```
INIT_WEIGHT = 100 # Initial weights of the edges
```

D. Initialization

The next step is initializing the key components of the simulation, which are the players and the relationship graph. Players are initialized by selecting from a predefined set of strategies, either randomized or fixed depending on the configured parameters. Each player is an instance of the chosen strategy, which are responsible for dictating how the player interacts with other player during the simulation.

```
# Define the players
if RANDOM_PLAYERS:
    players = [random.choice(strategies) for i in
range(PLAYER_COUNT)] # Randomize players
else:
    players = [
        axl.Cooperator(),
        axl.Defector(),
        axl.Defector(),
        axl.Cooperator(),
        axl.Cooperator(),
        axl.Defector(),
        axl.Cooperator(),
        axl.Cooperator(),
        axl.Cooperator(),
        axl.Defector(),
        axl.Defector()
    ]
```

Below is a list of the predefined strategies:

```
# List of strategies
strategies = [
    axl.Cooperator(),
    axl.Defector(),
    axl.TitForTat(),
    axl.Grudger(),
    axl.Random(),
    axl.Adaptive(),
    axl.AdaptiveTitForTat(),
    axl.Forgiver(),
    axl.ForgivingTitForTat(),
    axl.Bully(),
    axl.Grumpy(),
    axl.Punisher(),
    axl.Resurrection(),
    axl.Gradual(),
    axl.GradualKiller(),
    axl.CycleHunter(),
    axl.AntiTitForTat(),
    axl.Aggravater(),
    axl.HardTitForTat(),
    axl.HardGoByMajority(),
    axl.UsuallyCooperates(),
    axl.UsuallyDefects(),
    axl.SuspiciousTitForTat(),
    axl.WorseAndWorse(),
    axl.DoubleCrosser(),
    axl.Predator(),
    axl.Prober(),
    axl.NiceAverageCopier(),
    axl.CycleHunter(),
    axl.AntiCycler(),
    axl.EasyGo(),
    axl.OriginalGradual(),
    axl.Detective(),
    axl.NTitsForMTats(),
    axl.SneakyTitForTat(),
    axl.AverageCopier(),
    axl.WinStayLoseShift(),
    axl.WinShiftLoseStay()
]
```

The relationship player is modelled using an unweighted graph, which represents the players as nodes and relationship as weighted edges. The initial weight of each edge is set to a

predefined value, defined in the parameters as INIT_WEIGHT. The weight symbolizes the strength of the relationship between the two nodes it connects. This weight is not static and can be changed over time, depending on the interaction between the two adjacent players. The graph serves two purposes, for one it helps in tracking the dynamic and ever-evolving relationship between every pair of players, and it also provides a clear visualization of these relationships.

```
# Initialize the relationship graph
relationship_graph = nx.Graph()
for i, player in enumerate(players, start=1):
    relationship_graph.add_node(f"Player_{i}")

# Initialize graph edges
for i in range(len(players)):
    for j in range(i + 1, len(players)):
        relationship_graph.add_edge(f"Player_{i+1}",
f"Player_{j+1}", weight=INIT_WEIGHT)
```

Finally, a payoff accumulator is initialized to track the cumulative payoff for each player throughout simulation. It is implemented as a Python dictionary where the player acts as the key while their corresponding payoff acts as the value. The payoffs are updated after every round and subsequently displayed to allow for further analysis.

```
# Payoff accumulator for each player
payoff_accumulator = {f"Player_{i+1}": 0 for i in
range(len(players))}
```

E. Playing The Matches

The Axelrod library provides a built-in tournament system that facilitates large scale IPD simulations. Unfortunately, this built-in system does not fully meet the specific needs of this research. Specific components of this research such as the relationship dynamics through graph or the severance and rebuilding of relationship are not supported by Axelrod's tournament system. To overcome this limitation, a custom implementation of the tournament system was made. This allows greater control on how the tournament is played out and integration with the relationship graph.

After each match, the graph is dynamically updated to reflect the outcome of the interactions. This process is executed by the `update_relationships` function. If a relationship falls below the specified thresholds, then that relationship will be severed. This is handled by the `apply_thresholds` function. Once the thresholds are applied, the program identifies a severed relationship and evaluate the possibility of rebuilding it through the `rebuild_relationships` function.

```
def apply_thresholds(graph):
    # Sever relationship below the threshold

    edges_to_remove = [(u, v) for u, v, data in
graph.edges(data=True) if data['weight'] <= THRESHOLD]
    graph.remove_edges_from(edges_to_remove)

def rebuild_relationships(graph, players,
rebuild_chance=REBUILD_CHANCE):
    # Rebuild relationship based on random chances

    for i in range(len(players)):
        for j in range(i + 1, len(players)):
            if not graph.has_edge(f"Player_{i+1}",
f"Player_{j+1}"):

```

```
        if random.random() < rebuild_chance:
            graph.add_edge(f"Player_{i+1}",
f"Player_{j+1}", weight=INIT_WEIGHT // 2)

def update_relationships(graph, i, j, relationship_change):
    # Update the relationship after each round

    current_weight = graph.get_edge_data(f"Player_{i+1}",
f"Player_{j+1}", default={'weight': 0})['weight']
    graph.add_edge(f"Player_{i+1}", f"Player_{j+1}",
weight=current_weight + relationship_change)
```

The next step is to play out the matches. As mentioned before, the tournament uses a custom implementation to pit every player against every other players. This is done by using a nested loop to iterate through each pair of players. During the iteration, the program checks if there is an edge (relationship) between the players' nodes in the graph. If there is, the program proceeds to play a match with the number of turn specified in the simulation parameters. The payoffs are then counted for every interactions and the relationship is updated depending on both players' moves during the match.

```
def play_matches(players, graph):
    # Play matches based on the relationship graph

    results = {}
    for i, player_a in enumerate(players):
        for j in range(i + 1, len(players)):
            if graph.has_edge(f"Player_{i+1}", f"Player_{j+1}"):
                match = axl.Match((player_a, players[j]),
turns=NUM_TURNS)
                interactions = match.play()
                payoffs =
axl.interaction_utils.compute_final_score(interactions)
                a_payoff, b_payoff = int(payoffs[0]),
int(payoffs[1])
                results[(f"Player_{i+1}", f"Player_{j+1}")] =
(interactions, (a_payoff, b_payoff))

                payoff_accumulator[f"Player_{i+1}"] += a_payoff
                payoff_accumulator[f"Player_{j+1}"] += b_payoff

                relationship_change = 0
                for action_a, action_b in interactions:
                    if action_a == axl.Action.C and action_b ==
axl.Action.C:
                        relationship_change += 1
                    elif (action_a == axl.Action.C and action_b
== axl.Action.D) or (action_a == axl.Action.D and action_b ==
axl.Action.C):
                        relationship_change -= 1
                    elif action_a == axl.Action.D and action_b
== axl.Action.D:
                        relationship_change -= 2

                update_relationships(graph, i, j,
relationship_change)

    return results
```

Matches are then played out repeatedly in accordance to the number of rounds specified in the parameters. This is referred to as a *tournament*. The cumulative results from every matches decide the overall performance of the player. After a round has been played out, the program assesses the relationship between the players. If there are any relationships below the thresholds, then that relationships will be severed. Additionally, previously severed relationships may be rebuild, depending on the rebuild chance. If the Moran Process is toggled on, players with higher payoffs has a chance to reproduce and replace the strategy of another existing player.

```

for round_number in range(NUM_ROUNDS):
    print(f"Round {round_number + 1}")

    # Play matches
    match_results = play_matches(players, relationship_graph)

    # Apply thresholds
    apply_thresholds(relationship_graph)

    # Rebuild relationships
    rebuild_relationships(relationship_graph, players,
rebuild_chance=REBUILD_CHANCE)

    # Apply Moran process
    if USE_MORAN_PROCESS:
        moran_process(players, payoff_accumulator)

    # Print results after each round
    print(f"Total Payoffs and Strategies after Round
{round_number + 1}:")
    for i, player in enumerate(players):
        strategy_name = type(player).__name__
        total_payoff = payoff_accumulator[f"Player_{i+1}"]
        print(f"Player_{i+1}: {total_payoff} (Strategy:
{strategy_name})")

```

```

nx.draw(graph, pos, with_labels=True, node_color='skyblue',
edge_color='gray', font_size=10)
nx.draw_networkx_edge_labels(graph, pos,
edge_labels=edge_weights, font_size=10)
plt.show()

print("\nFinal Total Payoffs and Strategies after simulation:")
for i, player in enumerate(players):
    strategy_name = type(player).__name__
    total_payoff = payoff_accumulator[f"Player_{i+1}"]
    print(f"Player_{i+1}: {total_payoff} (Strategy:
{strategy_name})")

visualize_graph(relationship_graph)

```

IV. RESULT AND ANALYSIS

To begin testing the simulation, the initial focus would be on the two most basic strategies: cooperator, who always cooperate and defector, who always defect. This simple setup will provide a solid foundation for understanding the core dynamic of the simulation, such as how payoff accumulates and how relationships evolve. Below is the setup that will be used for this first experiment:

```

# Simulation parameters
THRESHOLD = 0 # Only allow matches if the weight is above this
threshold
REBUILD_CHANCE = 0.05 # Chance to rebuild severed
relationships
NUM_ROUNDS = 20 # Number of rounds in the simulation
NUM_TURNS = 20 # Number of turns each round between each player
USE_MORAN_PROCESS = False # Toggle for Moran process
RANDOM_PLAYERS = False # Toggle for randomized player
PLAYER_COUNT = 10 # Player count for randomize player
INIT_WEIGHT = 100 # Initial weights of the edges

```

The players will consist of five cooperators and five defectors.

```

def moran_process(players, payoffs):
    # Evolve strategies using the Moran process

    total_payoff = sum(payoffs.values())
    if total_payoff == 0:
        return # Avoid division by zero

    # Calculate selection probabilities based on payoffs
    selection_probs = [payoffs[f"Player_{i+1}"] / total_payoff
for i in range(len(players))]

    # Select a player to reproduce based on probabilities
    reproducing_index = random.choices(range(len(players)),
weights=selection_probs, k=1)[0]

    # Select a player to be replaced
    replaced_index = random.choice([i for i in
range(len(players)) if i != reproducing_index])

    # Replace the strategy of the selected player
    players[replaced_index] = type(players[reproducing_index])()

```

```

players = [
    axl.Cooperator(),
    axl.Defector(),
    axl.Defector(),
    axl.Cooperator(),
    axl.Cooperator(),
    axl.Defector(),
    axl.Cooperator(),
    axl.Cooperator(),
    axl.Defector(),
    axl.Defector()
]

```

To allow for easier analysis, the program will be modified to display the relationship graph after each round. Below are the results:

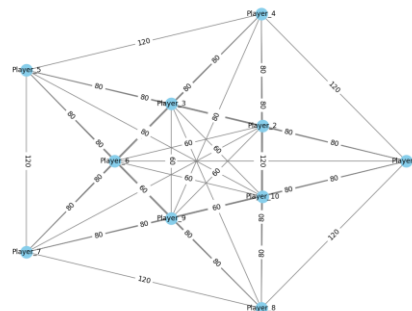


Fig. 4.1. Relationship Graph After Round 1
Source: Author

F. The Moran Process

The Axelrod library also provides a built-in mechanism to implement the Moran Process, but similar to tournament system, it does not account for the dynamic graph-based system implemented in this research. The Moran Process works by calculating the total accumulated payoffs of every players. Every players are then assigned weights based on their own accumulated payoffs in comparison to the total accumulated payoffs. Based on this weight, a random player will be selected to reproduce and replace the strategy of another player.

G. Visualization

Once the tournament is complete, the program will display the final payoff accumulated by each player. This provides insight into the overall performance of the player against every opponent. In addition to that, the program will visualize the state of the relationship graph after the final round, showcasing the end result of the evolving interactions between players.

```

def visualize_graph(graph):
    # Visualize the relationship graph

    pos = nx.kamada_kawai_layout(graph)
    edge_weights = nx.get_edge_attributes(graph, 'weight')

```

As shown by the graph, every nodes are still connected to each other meaning that no relationship have been severed. The difference lies in the weight of the edges. There are currently 3 distinct values for the weights of the edges which are 60, 80 and 120. The edges weighing 60 are clustered in the middle of the graphs and align with the defectors. Two defectors interacting with each other will cause the relationship to strain even faster than between a cooperator and defector. The edges weighing 80 indicates interaction between cooperator and defectors. This shows a reduction in relationship strength by 20 points which equals the number of turns within each round. The edges weighing 120 shows the interaction between cooperator. Different from the other two values, the weight actually increases from the initial weights, showcasing a fostering of trust between cooperator.

round. The relationship strength seems to change at the same pace and the accumulated payoff also shows similar growth.

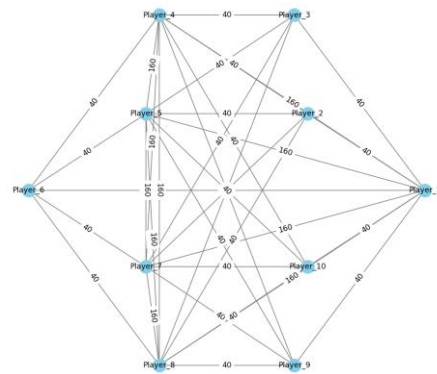


Fig. 4.3. Relationship Graph After Round 3
Source: Author

Things are starting to change in the third round. As shown in the graph, the defectors are no longer connected to each other. This means they won't further interact, causing them to lose potential payoff however miniscule it is. The accumulated payoff below still shows that the defectors are still leading in terms of overall performance.

Round 1
Total Payoffs and Strategies after Round 1:
Player_1: 240 (Strategy: Cooperator)
Player_2: 580 (Strategy: Defector)
Player_3: 580 (Strategy: Defector)
Player_4: 240 (Strategy: Cooperator)
Player_5: 240 (Strategy: Cooperator)
Player_6: 580 (Strategy: Defector)
Player_7: 240 (Strategy: Cooperator)
Player_8: 240 (Strategy: Cooperator)
Player_9: 580 (Strategy: Defector)
Player_10: 580 (Strategy: Defector)

Total Payoffs and Strategies after Round 3:
Player_1: 720 (Strategy: Cooperator)
Player_2: 1740 (Strategy: Defector)
Player_3: 1740 (Strategy: Defector)
Player_4: 720 (Strategy: Cooperator)
Player_5: 720 (Strategy: Cooperator)
Player_6: 1740 (Strategy: Defector)
Player_7: 720 (Strategy: Cooperator)
Player_8: 720 (Strategy: Cooperator)
Player_9: 1740 (Strategy: Defector)
Player_10: 1740 (Strategy: Defector)

The payoff accumulated from the first round showed that the defectors outperformed the cooperators by over twice as much. This outcome is consistent with well known Iterated Prisoner's Dilemma dynamics where defector gains greater immediate reward compared to cooperators.

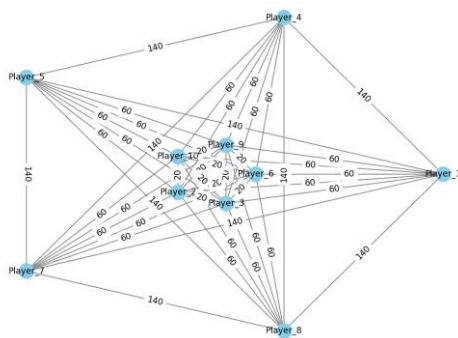


Fig. 4.2. Relationship Graph After Round 2
Source: Author

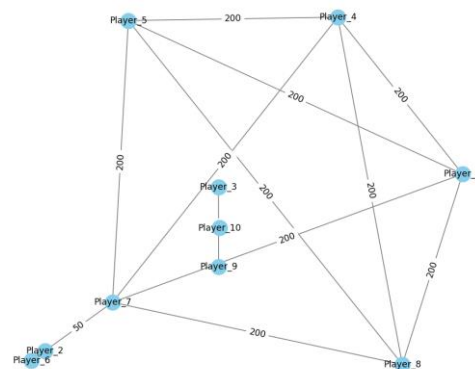


Fig. 4.4. Relationship Graph After Round 5
Source: Author

Fast forward to the fifth round, the shape of the graph has drastically changed. At this point, most relationships involving defector have been severed while the cooperators' still remain strong. Although, some severed relationships have been rebuilt.

Round 2
Total Payoffs and Strategies after Round 2:
Player_1: 480 (Strategy: Cooperator)
Player_2: 1160 (Strategy: Defector)
Player_3: 1160 (Strategy: Defector)
Player_4: 480 (Strategy: Cooperator)
Player_5: 480 (Strategy: Cooperator)
Player_6: 1160 (Strategy: Defector)
Player_7: 480 (Strategy: Cooperator)
Player_8: 480 (Strategy: Cooperator)
Player_9: 1160 (Strategy: Defector)
Player_10: 1160 (Strategy: Defector)

Total Payoffs and Strategies after Round 5:
Player_1: 1200 (Strategy: Cooperator)
Player_2: 2780 (Strategy: Defector)
Player_3: 2760 (Strategy: Defector)
Player_4: 1200 (Strategy: Cooperator)

The second round results are consistent with that of the first


```

Round 1
Total Payoffs and Strategies after Round 1:
Player_1: 297 (Strategy: Cooperator)
Player_2: 420 (Strategy: HardTitForTat)
Player_3: 379 (Strategy: UsuallyDefects)
Player_4: 449 (Strategy: Aggravater)
Player_5: 275 (Strategy: WorseAndWorse)
Player_6: 502 (Strategy: UsuallyDefects)
Player_7: 379 (Strategy: EasyGo)
Player_8: 498 (Strategy: UsuallyDefects)
Player_9: 454 (Strategy: Punisher)
Player_10: 400 (Strategy: WinStayLoseShift)

```

```

Final Total Payoffs and Strategies after simulation:
Player_1: 6623 (Strategy: WinStayLoseShift)
Player_2: 6400 (Strategy: HardTitForTat)
Player_3: 3098 (Strategy: HardTitForTat)
Player_4: 3569 (Strategy: HardTitForTat)
Player_5: 6152 (Strategy: HardTitForTat)
Player_6: 2556 (Strategy: UsuallyDefects)
Player_7: 6326 (Strategy: HardTitForTat)
Player_8: 2377 (Strategy: UsuallyDefects)
Player_9: 6791 (Strategy: WinStayLoseShift)
Player_10: 6383 (Strategy: HardTitForTat)

```

Both of these examples illustrate that adaptive cooperative strategies win in the long run, while extremely aggressive strategies peak early and find it difficult to maintain relationship in the long run.

V. CONCLUSION

The result of the simulation demonstrates the dynamic nature of strategies in the Iterated Prisoner's Dilemma. Aggressive strategies tend to achieve higher payoff early in the simulation but receive a sharp decline as relationship dynamics evolve and ties are severed. In contrast, a more cooperative strategy have a slow start but shows more long term success. A successful strategy must strike a balance between the two. These strategies thrive off being able to foster cooperation and adapt to the behavior of others.

The usage of weighted graphs has managed to effectively model relationship dynamics between multiple players, adding an additional layer of complexity to the traditional Iterated Prisoner's Dilemma. While it has managed to model relationships within a vacuum, it does not provide the full picture. It is not the only tool available. Regardless, the Prisoner's Dilemma has proven to be a powerful tool to analyze relationships. Whether that is between individuals, organization or even nations, there is always an application of the PD there. While the temptation to defect may be high, cooperation and the ability to adapt proved to be more sustainable and leads to better outcome.

VI. APPENDIX

Full source code of the program is available at:
https://github.com/DanielDPW/Makalah_IF1220_Matdis

VII. ACKNOWLEDGMENT

The author expresses gratitude to lecturer Dr. Rinaldi Munir, M. T. as the lecturer of IF1220 Discrete Mathematics course, for his guidance and resource provided in finishing this paper. The author also expresses thanks to Robert Axelrod as one of the

main reference and to the people behind the Axelrod Library for the implementation of this project.

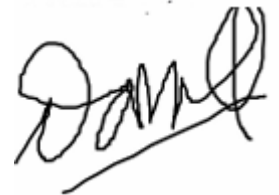
REFERENCES

- [1] R. Munir. "Graf (Bagian 1)", 2024. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf> (accessed: Jan. 4, 2025).
- [2] R. Munir. "Graf (Bagian 2)", 2024. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf> (accessed: Jan. 4, 2025).
- [3] R. Munir. "Graf (Bagian 3)", 2024. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf> (accessed: Jan. 4, 2025).
- [4] K. H. Rosen, *Discrete Mathematics and Its Applications*, 7th ed. New York, NY, USA: McGraw-Hill, 2012.
- [5] S. Kuhn, "Prisoner's Dilemma," *The Stanford Encyclopedia of Philosophy* (Winter 2024 Edition), E. N. Zalta & U. Nodelman (eds.), <https://plato.stanford.edu/archives/win2024/entries/prisoner-dilemma/> (accessed: Jan. 6, 2025).
- [6] R. Axelrod, *The Evolution of Cooperation*, 1st ed. New York, NY, USA: Basic Books, 1984
- [7] *Axelrod Documentation*, <https://axelrod.readthedocs.io/en/stable/> (accessed: Jan. 6, 2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Januari 2025



Daniel Pedrosa Wu 13523099