A Graph-Based Optimization Framework on Pokémon VGC Team Selection in Regulation G

Nicholas Wise Saragih Sumbayak - 13524037
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nicholasaragih@gmail.com, 13524037@std.stei.itb.ac.id

***Abstract*—This paper applies a use case of graph theory to simulate a more concise and quantitative approach on team selection in Pokemon VGC. The result is a framework that uses a bipartite graph to map matchups between Pokemon and a complete 6 node graph to model the relationships between Pokemon in a team. The heaviest subgraph consisting of four nodes will be denoted as the strongest team against the opponent's team based on this framework**

***Keywords*—Graph Theory, Pokemon VGC, Team Synergy, Heaviest Subgraph Problem**

I. Introduction

In the Pokémon community, one of the most prevalent sub-communities is the Competitive Pokémon community, building off the well-established battle system introduced in mainstream Pokémon titles. There are several ways to play Pokémon competitively, all differentiated based on their battle formats, list of legal Pokémon for such format, and even which generation of Pokémon's battle rules to adopt. This paper focuses on Generation 9 Pokémon VGC Regulation G, the latest completed battle regulation during the time this paper was written.

The format officially supported by The Pokémon Company™, otherwise known as Pokémon VGC (Video Game Competitions) consists of a double battle, where each player picks a pair of Pokémon to simultaneously fight the other's and is subsequently much faster-paced compared to a traditional single battle. Before each battle, a team preview is initiated. Both players get to look at each other's team – a chance to evaluate and strategize the gameplan moving forward – and pick only 4 of 6 for battle, which means the moment the battle starts both teams have already shown half of their chosen Pokémon.

The nature of the selection of those 4 Pokémon introduces a high-stakes combinatorial decision: from a pool of 6 Pokémon, there are a combined total of $\binom{6}{4} = 15$ valid combinations to choose from, each with their own distinct synergy profiles and matchup strengths. The most optimal team to use is usually determined purely by intuition and experience, but this approach can very often overlook possible powerful combinations from either team. In an attempt to determine which combination is the strongest from a synergy and matchup perspective, a graph-theoretic framework that models both internal team synergy and individual matchup potential. The framework uses 2 distinct graph models:

- A bipartite undirected weighted 6-to-6 graph to model each individual matchup of Pokémon and determine the matchup score against such team.
- An undirected weighted 6-node complete graph – otherwise known as a K_6 graph – to model team synergy, where each node represents each Pokémon and how strong that Pokémon matches up against the opponent's team represented by it's weight and each edge weight quantifies pairwise synergy.

By leveraging these graph structures, the proposed method aims to identify the most optimal subset of four Pokémon that maximizes internal synergy and external matchup advantage.

II. Theoretical Foundation

A. Graph Theory

In graph theory, a graph is a mathematical structure used to model pairwise relationships between multiple objects consisting of vertices and edges.

1) Definition

A graph G is defined as $G = (V, E)$, in which V represents a nonempty set of vertices and E represents a set of edges[1].

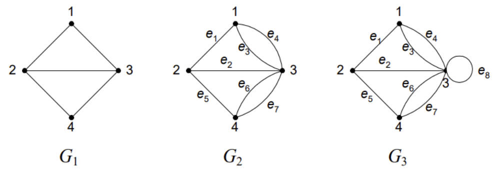


Figure 2.1 (a) a simple graph, (b) a multi-graph, (c) a pseudo-graph
Source: Rinaldi Munir Homepage

In figures 2.1, each graph is defined as follows:

- $G_1 = (V_1, E_1)$, $V_1 = \{1, 2, 3, 4\}$, $E_1 = \{(1,2), (1,3), (2,3), (2,4), (3,4)\}$

- $G_2 = (V_2, E_2)$, $V_2 = \{1, 2, 3, 4\}$, $E_2 = \{(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4)\} = \{e1, e2, e3, e4, e5, e6, e7\}$
- $G_3 = (V_3, E_3)$, $V_3, = \{1, 2, 3, 4\}$, $E_3 = \{(1,2), (2,3), (1,3), (1,3), (2,4), (3,4), (3,4), (3,3)\} = \{e1, e2, e3, e4, e5, e6, e7, e8\}$

2) Terminologies

Moving forward, several terminologies in graph theory are critical to understand.

- Adjacence

  Any two vertices are considered adjacent if both are directly connected to each other by an edge.
- Incidence

  For any edge $e = (v_j, v_k)$, e is incident with v_j and v_k, or in simpler terms e must connect vertices v_j and v_k.
- Degree

  The degree $d(v)$ for vertex v is determined by the amount of vertices adjacent to such vertex.

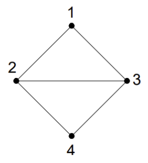


Figure 2.2 Example of a graph
Source: Rinaldi Munir Homepage

Let the graph above be $G = (V, E)$, with $V = \{1,2,3,4\}$ and $E = \{(1,2), (1,3), (2,3), (2,3), (3,4)\}$. 1 is adjacent with 2 and 3, (1,2) is incident with 1 and 2, inferred from the illustration above and it's notation. The degrees for each vertex are determined as follows: $d(1) = d(4) = 2$, $d(2) = d(3) = 3$.

- Subgraph

  Let $G = (V, E)$ be a graph. $G_1 = (V_1, E_1)$ is considered a subgraph of G if $V_1 \subseteq$ V and $E_1 \subseteq$ E. In other words, the vertices and edges in subgraph G_1 must be a subset of those in graph G.

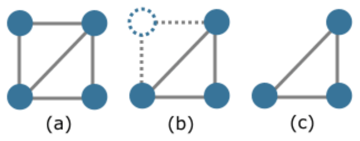


Figure 2.3 (c) is a subgraph of (a)
Source: https://symbio6.nl/en/blog/theory/definition/subgraph

- Weighted Graphs

  A graph is considered a weighted graph if every edge contains a "weight" or value. This value may represent many things e.g. the distance between 2 points on a map.

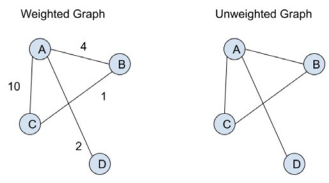


Figure 2.4 Comparison between a weighted and an unweighted graph
Source: Rinaldi Munir Homepage

- Complete Graphs

  Let G be a graph. G is considered a complete graph if every vertex is adjacent to every other vertex in G. A complete graph with n vertices is commonly written as K_n. Each vertex in a complete graph must have a degree of $n - 1$ and the total amount of edges in a complete graph can be calculated with the formula $\frac{n(n-1)}{2}$.

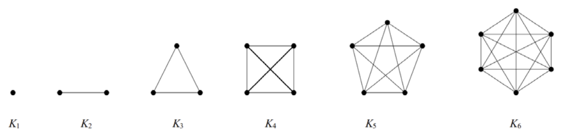


Figure 2.5 Complete graphs K_n ranging from $n = 1$ to $n = 6$
Source: Rinaldi Munir Homepage

- Bipartite Graphs

  Let $G = (V, E)$ be a graph. If V can be separated in two separate subsets V_1 and V_2 such that for every edge $e \in E$ connects a vertex in V_1 to a vertex in V_2, G is considered a bipartite graph and may be denoted as $G(V_1, V_2)$.

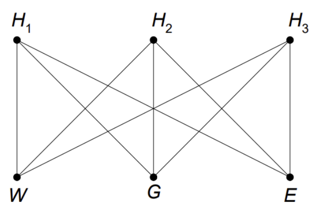


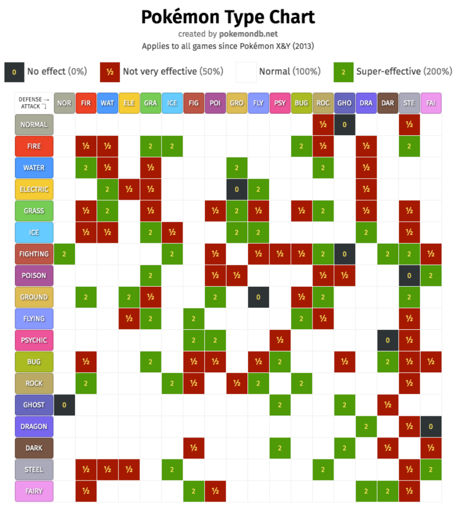
Figure 2.6 Example of a bipartite graph, $V_1 = \{H_1, H_2, H_3\}$ and $V_2 = \{W, G, E\}$
Source: Rinaldi Munir Homepage

B. *Competitive Pokémon*

Pokémon itself might be a familiar term for a majority of people. The way Pokémon battles work might seem simple at first, but in competitive battling, you must put every single aspect of battling into consideration, which gets very confusing very quickly.

1) *Pokémon Type System*

In Pokémon, each Pokémon has either one or two types (e.g. Fire, Water, Ground-Rock), which define how it interacts with other types through type-effectiveness. Every move in Pokémon also has a type. The effectiveness of a move against a Pokémon is determined by the relationship between the move's type and the Pokémon's type. For example, a water type move is considered "super-effective" against a fire type Pokémon, therefore dealing double the damage it would otherwise. If a Pokémon has two types, the effectiveness of a move dealt to that Pokémon will be the product of effectiveness of the move against each type. The "type-effectiveness" chart is as follows.



Pokémon Type Chart

created by pokemondb.net

Applies to all games since Pokémon X&Y (2013)

0 No effect (0%) ½ Not very effective (50%) Normal (100%) 2 Super-effective (200%)

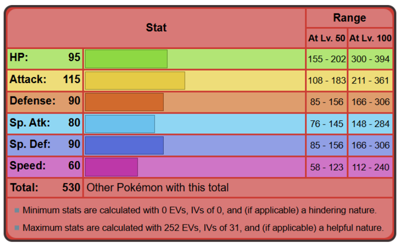
DEFENSE → ATTACK ↴	NOR	FIR	WAT	ELE	GRA	ICE	FIG	POI	GRO	FLY	PSY	BUG	ROC	GHO	DRA	DAR	STE	FAI
NORMAL													½	0			½	
FIRE		½	½		2	2						2	½		½		2	
WATER		2	½		½				2				2		½			
ELECTRIC			2	½	½				0	2					½			
GRASS		½	2		½			½	2	½		½	2		½		½	
ICE		½	½		2	½			2	2					2		½	
FIGHTING	2					2		½		½	½	½	2	0		2	2	½
POISON					2			½	½				½	½			0	2
GROUND		2		2	½			2		0		½	2				2	
FLYING				½	2		2					2	½				½	
PSYCHIC							2	2			½					0	½	
BUG		½			2		½	½		½	2			½		2	½	½
ROCK		2				2	½		½	2		2					½	
GHOST	0										2			2		½		
DRAGON															2		½	0
DARK							½				2			2		½		½
STEEL		½	½	½		2							2				½	2
FAIRY		½					2	½							2	2	½	

Figure 3.1 The Pokémon Type Chart, the rows representing the move type (attacking) and the columns representing the Pokémon type (defending).

Source: https://pokemondb.net/type

2) *Pokémon Base Stats (Statistics)*

Each individual Pokémon species has their own base "stats" that describe their capabilities, consisting of 6 dimensions: HP (Hit Points) determines how many hitpoints a Pokémon has by default, Attack and Special Attack determine how much damage a Pokémon deals using a physical moves and special moves respectively, Defense and Special Defense help mitigate the damage received from physical and special moves, and Speed determines the order in which each Pokémon moves in. These base stats are modified by EVs (Effort Values) and IVs (Individual Values), but for modeling purposes, a Pokémon's capability in a certain category are mainly represented by their base stats.



Stat			Range	
			At Lv. 50	At Lv. 100
HP:	95		155 - 202	300 - 394
Attack:	115		108 - 183	211 - 361
Defense:	90		85 - 156	166 - 306
Sp. Atk:	80		76 - 145	148 - 284
Sp. Def:	90		85 - 156	166 - 306
Speed:	60		58 - 123	112 - 240
Total:	530	Other Pokémon with this total		

Minimum stats are calculated with 0 EVs, IVs of 0, and (if applicable) a hindering nature.

Maximum stats are calculated with 252 EVs, IVs of 31, and (if applicable) a helpful nature.

Figure 3.2 Incineroar's base stats, with the range determining the possible range of a Pokémon's actual stats based on IVs and EVs

Source: https://bulbapedia.bulbagarden.net/wiki/Incineroar

3) *Pokémon VGC Battle Format*

As explained before, the VGC battle format consists of a double battle, where each player must have two active Pokémon on the battlefield if possible. Players are given full-visibility of each other's team, but not the four Pokémon selected to battle. After both teams choose their four Pokémon, the battle is initiated as a turn-based battle, just like a traditional Pokémon single battle, but this time involving four Pokémon and any Pokémon being able to target any other Pokémon on the field, even it's own teammate[2].

Figure 3.3 A double battle in Pokémon Scarlet

Source: https://victoryroad.pro/sv-rental-teams-reg-set-a/

Each Pokémon may know up to 4 moves, whether it be an attack or a status move, that are deemed legal for the format. A Pokémon may also hold a "held item" that may influence it's utility. For example, a Choice Scarf multiplies it's holder's speed by 1.5x, but it may only choose one move to use until it switches out. Finally, each Pokémon can have an ability, where each species usually has 1-3 abilities to choose from. Abilities are also essential in determining a Pokémon's role and capabilities in battle, as it can turn a Pokémon that is otherwise weak into a viable Pokémon, and a good Pokémon into an amazing one. For example, Torkoal may seem like a rather weak Pokémon solely from it's stats, but it's ability Drought is a staple in "Sun" teams as it sets the "Sunny" weather as it switches in, powering up fire types and activating abilities such as Chlorophyll.

III. Implementation

The implementation of this proposed framework works by initiating two graphs, as mentioned in the introduction. The framework will focus on two main aspects in team selection, which are team synergy and matchup scoring. For both graphs, every vertex will represent a single Pokémon, but the edges will represent different things. In the team synergy graph, each edge will have a weight representing how well each Pokémon complement each other. In the matchup graph, each edge will represent how strong each Pokémon is against the opposing team's Pokémon. The method of finding the strongest combination will involve finding the strongest combination of weights between both graphs.

A. Framework Foundation

The fundamental idea in this framework is to quantify each Pokémon's performance both against the opponent's team and with it's own team. To start off, each Pokémon's matchup score will be determined using a bipartite 6-by-6 graph, otherwise known as a K_6 graph. The graph will be split into two sets of vertices: V_1 representing the player's team and each vertex being mapped to every vertex in V_2, and each edge weight showing how well each Pokémon in the player's does against the opponents. The weights will be measured based on how well a player's Pokémon hits the enemy's, how weak it is against the enemy's Pokémon, and how their abilities match up.

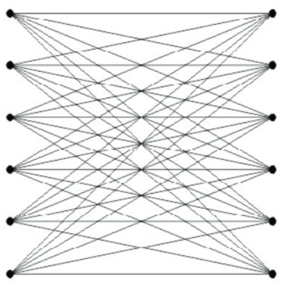


Figure 3.1 A $K_{6,6}$ Bipartite Graph

Source: https://www.researchgate.net/figure/a-The-complete-bipartite-graph-K6-6-and-b-the-line-graph-of-K6-6_fig4_356817294

After determining the weight of each edge, the weights will be averaged to determine the overall strength of the Pokémon against the enemy team. This average score will be used in the as the weights of the vertices in the team synergy graph.

In the team synergy graph, every Pokémon will be adjacent to each other, consequently forming a complete 6 node graph, otherwise known as a K_6 graph. In this graph, each edge will have a weight determining how well each Pokémon complement each other based on their typing and abilities.

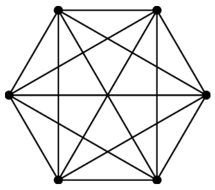


Figure 3.2 A K_6 Graph

Source: https://www.researchgate.net/figure/The-complete-graph-K-6_fig1_318598613

After the weight of each node and edge has been determined, all combinations of nodes will be checked to find the strongest combination using a brute-force method. Due to the nature of the amount vertices in the graph always being fixed, a brute-force approach won't have severe consequences in computing the strongest combination.

B. Programming Setup

To build this framework, Python was chosen for it's extensive library selection and enhanced capabilities in forming and illustrating graphs. The most important libraries in building this project include but are not limited to:

- *poke_env*: A Python environment equipped with multiple APIs to help simulate, parse, and analyze Pokémon Showdown battles, an online Pokémon battle simulator. It provides several tools to be able to parse and interpret Pokémon data.
- *networkx:* A powerful graph-based library essential to this framework, capable of constructing both the synergy and matchup graph, as well as performing graph-based computations at high efficiency.
- *matplotlib*: A 2D plotting library that is used for visualizing data. In this framework, it is used to visualize the graphs formed and make the framework much more readable.
- *pickle*: In order to save Pokémon data across files, this library is used due to the high reusability factor it adds.

C. Pokémon Data Parsing

In order to use this framework, the information of both Pokémon teams must be acquired. The framework will receive inputs of Pokémon teams in the form of Pokémon Showdown's team builder format that is widely used in the community. The format will include the Pokémon's name, item, ability, moves, and other critical Pokémon information. The following is an example of a Pokémon written in such format.

```
Miraidon @ Choice Specs

Ability: Hadron Engine

Level: 50

Tera Type: Fairy
```

EVs: 44 HP / 4 Def / 244 SpA / 12 SpD / 204 Spe

Modest Nature

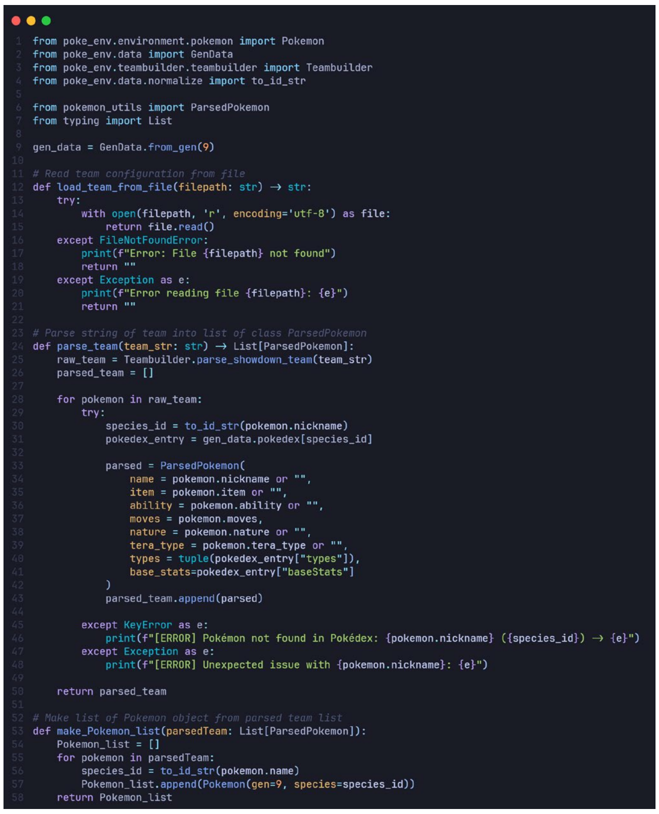
- Electro Drift

- Draco Meteor

- Volt Switch

- Dazzling Gleam

Each team will have 6 Pokémon, and subsequently each team will consist of 6 of these Pokémon in team builder format. These Pokémon will be parsed into a class named ParsedPokemon for easier data access.

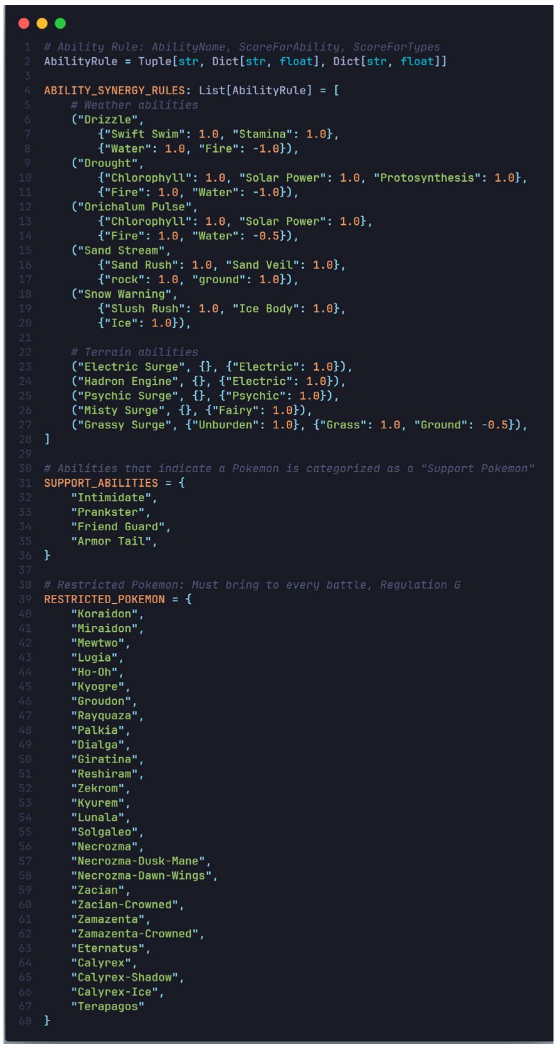


```python
from poke_env.environment.pokemon import Pokemon
from poke_env.data import GenData
from poke_env.teambuilder.teambuilder import Teambuilder
from poke_env.data.normalize import to_id_str

from pokemon_utils import ParsedPokemon
from typing import List

gen_data = GenData.from_gen(9)

# Read team configuration from file
def load_team_from_file(filepath: str) → str:
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: File {filepath} not found")
        return ""
    except Exception as e:
        print(f"Error reading file {filepath}: {e}")
        return ""

# Parse string of team into list of class ParsedPokemon
def parse_team(team_str: str) → List[ParsedPokemon]:
    raw_team = Teambuilder.parse_showdown_team(team_str)
    parsed_team = []

    for pokemon in raw_team:
        try:
            species_id = to_id_str(pokemon.nickname)
            pokedex_entry = gen_data.pokedex[species_id]

            parsed = ParsedPokemon(
                name = pokemon.nickname or "",
                item = pokemon.item or "",
                ability = pokemon.ability or "",
                moves = pokemon.moves,
                nature = pokemon.nature or "",
                tera_type = pokemon.tera_type or "",
                types = tuple(pokedex_entry["types"]),
                base_stats=pokedex_entry["baseStats"]
            )
            parsed_team.append(parsed)

        except KeyError as e:
            print(f"[ERROR] Pokémon not found in Pokédex: {pokemon.nickname} ({species_id}) → {e}")
        except Exception as e:
            print(f"[ERROR] Unexpected issue with {pokemon.nickname}: {e}")

    return parsed_team

# Make list of Pokemon object from parsed team list
def make_Pokemon_list(parsedTeam: List[ParsedPokemon]):
    Pokemon_list = []
    for pokemon in parsedTeam:
        species_id = to_id_str(pokemon.name)
        Pokemon_list.append(Pokemon(gen=9, species=species_id))
    return Pokemon_list
```

Figure 3.3 Showdown Format Parser
Source: https://github.com/nicholaswisee/Makalah-Matdis

D. Synergy Calculations

The synergy score between Pokémon will be calculated using 3 aspects, which are defensive synergy, offensive synergy, and ability synergy. The defensive synergy between two Pokémon is calculated by determining how many types a Pokémon resists that would otherwise be a weakness of the other Pokémon. For the offensive synergy, it is calculated by determining how many types in total both Pokémon can hit for super effective damage. Lastly, the ability synergy is calculated using a special datatype for this specific use case. This datatype is named AbilityRule, which is used to map the relationship between certain important abilities and other abilities and types that would be affected by those abilities. Following that, some dictionaries were made to be categorize certain specific abilities and Pokémon. The support abilities list was made to classify certain support abilities that would always give positive synergy to teammates, and the restricted Pokémon list would always have strong synergy with the team regardless, as they are almost

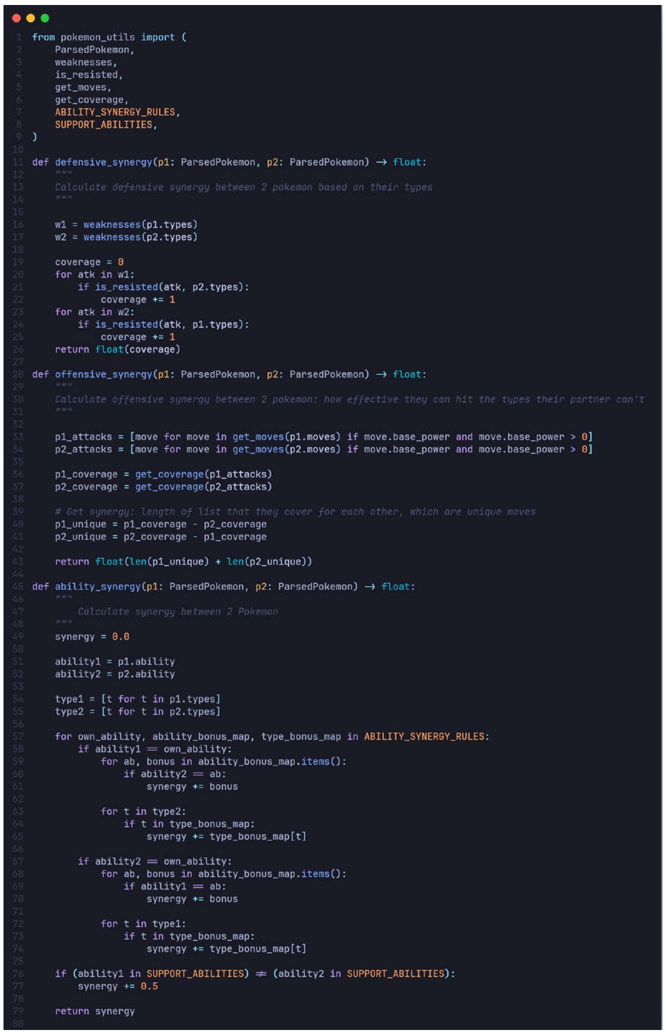


```python
# Ability Rule: AbilityName, ScoreForAbility, ScoreForTypes
AbilityRule = Tuple[str, Dict[str, float], Dict[str, float]]

ABILITY_SYNERGY_RULES: List[AbilityRule] = [
    # Weather abilities
    ("Drizzle",
        {"Swift Swim": 1.0, "Stamina": 1.0},
        {"Water": 1.0, "Fire": -1.0}),
    ("Drought",
        {"Chlorophyll": 1.0, "Solar Power": 1.0, "Protosynthesis": 1.0},
        {"Fire": 1.0, "Water": -1.0}),
    ("Orichalum Pulse",
        {"Chlorophyll": 1.0, "Solar Power": 1.0},
        {"Fire": 1.0, "Water": -0.5}),
    ("Sand Stream",
        {"Sand Rush": 1.0, "Sand Veil": 1.0},
        {"rock": 1.0, "ground": 1.0}),
    ("Snow Warning",
        {"Slush Rush": 1.0, "Ice Body": 1.0},
        {"Ice": 1.0}),

    # Terrain abilities
    ("Electric Surge", {}, {"Electric": 1.0}),
    ("Hadron Engine", {}, {"Electric": 1.0}),
    ("Psychic Surge", {}, {"Psychic": 1.0}),
    ("Misty Surge", {}, {"Fairy": 1.0}),
    ("Grassy Surge", {"Unburden": 1.0}, {"Grass": 1.0, "Ground": -0.5}),
]

# Abilities that indicate a Pokemon is categorized as a "Support Pokemon"
SUPPORT_ABILITIES = {
    "Intimidate",
    "Prankster",
    "Friend Guard",
    "Armor Tail",
}

# Restricted Pokemon: Must bring to every battle, Regulation G
RESTRICTED_POKEMON = {
    "Koraidon",
    "Miraidon",
    "Mewtwo",
    "Lugia",
    "Ho-Oh",
    "Kyogre",
    "Groudon",
    "Rayquaza",
    "Palkia",
    "Dialga",
    "Giratina",
    "Reshiram",
    "Zekrom",
    "Kyurem",
    "Lunala",
    "Solgaleo",
    "Necrozma",
    "Necrozma-Dusk-Mane",
    "Necrozma-Dawn-Wings",
    "Zacian",
    "Zacian-Crowned",
    "Zamazenta",
    "Zamazenta-Crowned",
    "Eternatus",
    "Calyrex",
    "Calyrex-Shadow",
    "Calyrex-Ice",
    "Terapagos"
}
```

always the strongest Pokémon on the team.

Figure 3.4 Ability Synergy Rules Map
Source: https://github.com/nicholaswisee/Makalah-Matdis

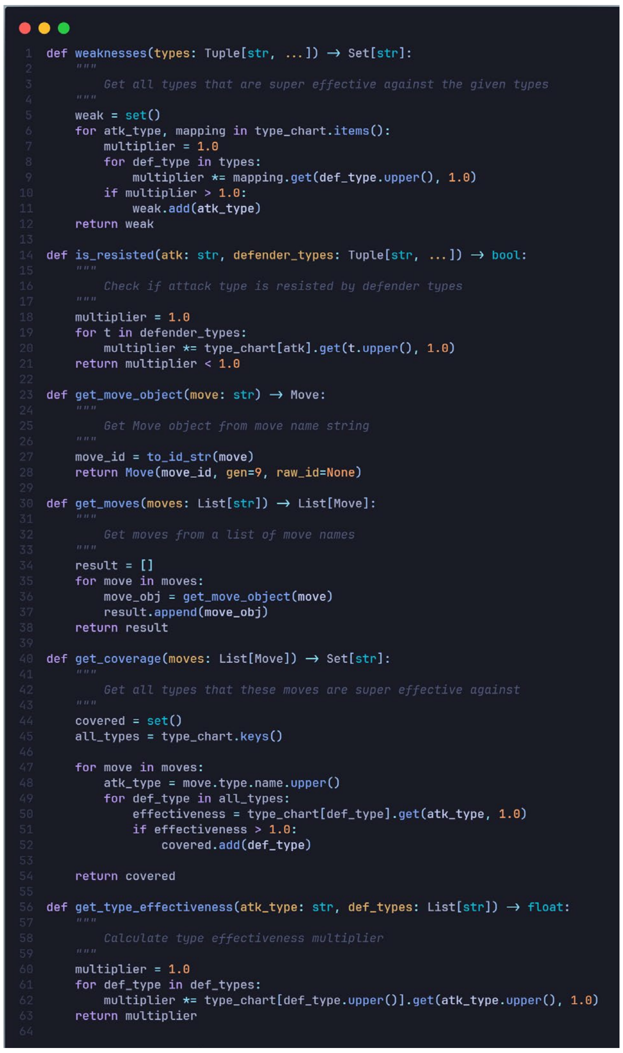
The given calculations for defensive synergy, offensive synergy, and ability synergy are done in the following.



```python
from pokemon_utils import (
    ParsedPokemon,
    weaknesses,
    is_resisted,
    get_moves,
    get_coverage,
    ABILITY_SYNERGY_RULES,
    SUPPORT_ABILITIES,
)

def defensive_synergy(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
    Calculate defensive synergy between 2 pokemon based on their types
    """

    w1 = weaknesses(p1.types)
    w2 = weaknesses(p2.types)

    coverage = 0
    for atk in w1:
        if is_resisted(atk, p2.types):
            coverage += 1
    for atk in w2:
        if is_resisted(atk, p1.types):
            coverage += 1
    return float(coverage)

def offensive_synergy(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
    Calculate offensive synergy between 2 pokemon: how effective they can hit the types their partner can't
    """

    p1_attacks = [move for move in get_moves(p1.moves) if move.base_power and move.base_power > 0]
    p2_attacks = [move for move in get_moves(p2.moves) if move.base_power and move.base_power > 0]

    p1_coverage = get_coverage(p1_attacks)
    p2_coverage = get_coverage(p2_attacks)

    # Get synergy: length of list that they cover for each other, which are unique moves
    p1_unique = p1_coverage - p2_coverage
    p2_unique = p2_coverage - p1_coverage

    return float(len(p1_unique) + len(p2_unique))

def ability_synergy(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
        Calculate synergy between 2 Pokemon
    """
    synergy = 0.0

    ability1 = p1.ability
    ability2 = p2.ability

    type1 = [t for t in p1.types]
    type2 = [t for t in p2.types]

    for own_ability, ability_bonus_map, type_bonus_map in ABILITY_SYNERGY_RULES:
        if ability1 == own_ability:
            for ab, bonus in ability_bonus_map.items():
                if ability2 == ab:
                    synergy += bonus

            for t in type2:
                if t in type_bonus_map:
                    synergy += type_bonus_map[t]

        if ability2 == own_ability:
            for ab, bonus in ability_bonus_map.items():
                if ability1 == ab:
                    synergy += bonus

            for t in type1:
                if t in type_bonus_map:
                    synergy += type_bonus_map[t]

    if (ability1 in SUPPORT_ABILITIES) != (ability2 in SUPPORT_ABILITIES):
        synergy += 0.5

    return synergy
```

Figure 3.5 Team Synergy Calculations
Source: https://github.com/nicholaswisee/Makalah-Matdis

In making these functions, some helper functions are also used to make data retrieval much more convenient, such as getting type effectiveness, making new objects, and determining the weaknesses of a Pokémon. Inside this file, there are also certain ability rules used to determine the relationship between an ability and other abilities or types. Since this framework is based on Regulation G, the list of restricted Pokémon are also included and are used to give bonus points to such Pokémon, as each team is limited to only one restricted Pokémon and are usually the center of each team.

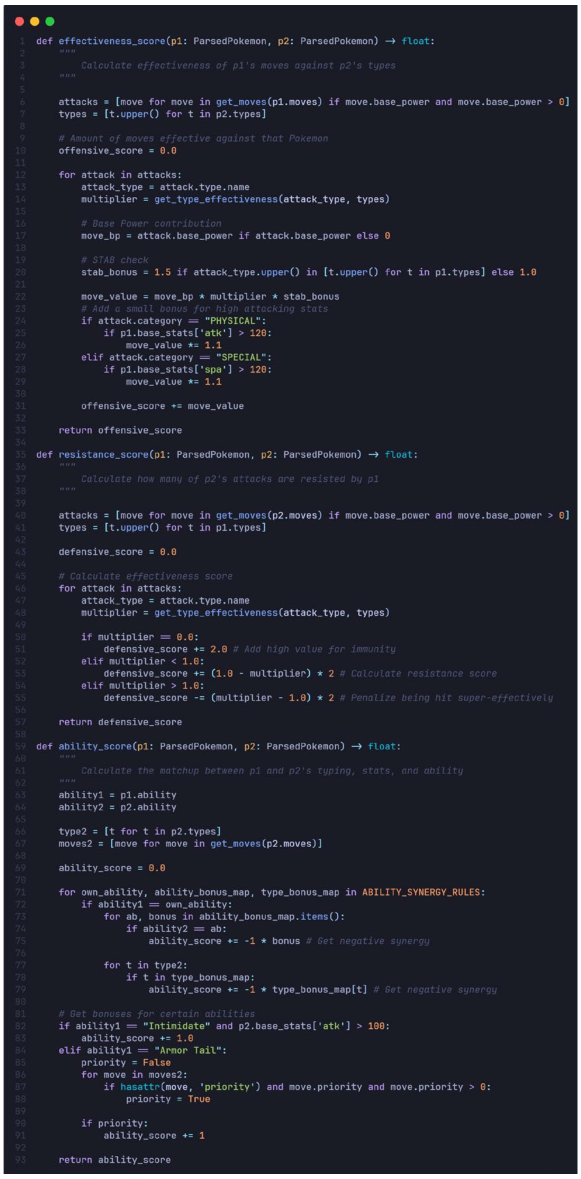


```python
def weaknesses(types: Tuple[str, ...]) -> Set[str]:
    """
        Get all types that are super effective against the given types
    """
    weak = set()
    for atk_type, mapping in type_chart.items():
        multiplier = 1.0
        for def_type in types:
            multiplier *= mapping.get(def_type.upper(), 1.0)
        if multiplier > 1.0:
            weak.add(atk_type)
    return weak

def is_resisted(atk: str, defender_types: Tuple[str, ...]) -> bool:
    """
        Check if attack type is resisted by defender types
    """
    multiplier = 1.0
    for t in defender_types:
        multiplier *= type_chart[atk].get(t.upper(), 1.0)
    return multiplier < 1.0

def get_move_object(move: str) -> Move:
    """
        Get Move object from move name string
    """
    move_id = to_id_str(move)
    return Move(move_id, gen=9, raw_id=None)

def get_moves(moves: List[str]) -> List[Move]:
    """
        Get moves from a list of move names
    """
    result = []
    for move in moves:
        move_obj = get_move_object(move)
        result.append(move_obj)
    return result

def get_coverage(moves: List[Move]) -> Set[str]:
    """
        Get all types that these moves are super effective against
    """
    covered = set()
    all_types = type_chart.keys()

    for move in moves:
        atk_type = move.type.name.upper()
        for def_type in all_types:
            effectiveness = type_chart[def_type].get(atk_type, 1.0)
            if effectiveness > 1.0:
                covered.add(def_type)

    return covered

def get_type_effectiveness(atk_type: str, def_types: List[str]) -> float:
    """
        Calculate type effectiveness multiplier
    """
    multiplier = 1.0
    for def_type in def_types:
        multiplier *= type_chart[def_type.upper()].get(atk_type.upper(), 1.0)
    return multiplier
```

Figure 3.6 Helper Functions
Source: https://github.com/nicholaswisee/Makalah-Matdis

E. Matchup Calculations

Determining the matchup score between the team's Pokémon and the opponent's Pokémon involves calculating the effectiveness score, weakness score, and ability score between the Pokémon on each team. These calculations are done through an approach similar to the team synergy calculations, but this time it is much more thorough, as the matchup is the most deciding factor when selecting a team. The calculations are down through the following functions.



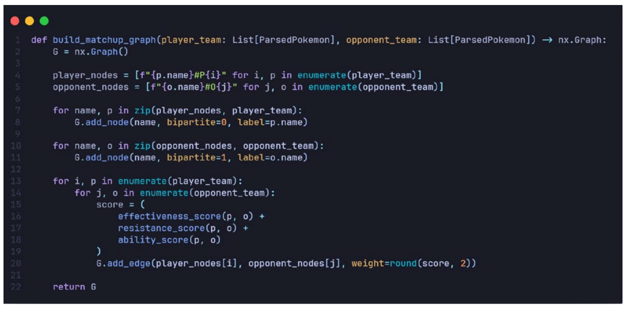
```python
def effectiveness_score(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
        Calculate effectiveness of p1's moves against p2's types
    """

    attacks = [move for move in get_moves(p1.moves) if move.base_power and move.base_power > 0]
    types = [t.upper() for t in p2.types]

    # Amount of moves effective against that Pokemon
    offensive_score = 0.0

    for attack in attacks:
        attack_type = attack.type.name
        multiplier = get_type_effectiveness(attack_type, types)

        # Base Power contribution
        move_bp = attack.base_power if attack.base_power else 0

        # STAB check
        stab_bonus = 1.5 if attack_type.upper() in [t.upper() for t in p1.types] else 1.0

        move_value = move_bp * multiplier * stab_bonus
        # Add a small bonus for high attacking stats
        if attack.category == "PHYSICAL":
            if p1.base_stats['atk'] > 120:
                move_value *= 1.1
        elif attack.category == "SPECIAL":
            if p1.base_stats['spa'] > 120:
                move_value *= 1.1

        offensive_score += move_value

    return offensive_score

def resistance_score(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
        Calculate how many of p2's attacks are resisted by p1
    """

    attacks = [move for move in get_moves(p2.moves) if move.base_power and move.base_power > 0]
    types = [t.upper() for t in p1.types]

    defensive_score = 0.0

    # Calculate effectiveness score
    for attack in attacks:
        attack_type = attack.type.name
        multiplier = get_type_effectiveness(attack_type, types)

        if multiplier == 0.0:
            defensive_score += 2.0 # Add high value for immunity
        elif multiplier < 1.0:
            defensive_score += (1.0 - multiplier) * 2 # Calculate resistance score
        elif multiplier > 1.0:
            defensive_score -= (multiplier - 1.0) * 2 # Penalize being hit super-effectively

    return defensive_score

def ability_score(p1: ParsedPokemon, p2: ParsedPokemon) -> float:
    """
        Calculate the matchup between p1 and p2's typing, stats, and ability
    """
    ability1 = p1.ability
    ability2 = p2.ability

    type2 = [t for t in p2.types]
    moves2 = [move for move in get_moves(p2.moves)]

    ability_score = 0.0

    for own_ability, ability_bonus_map, type_bonus_map in ABILITY_SYNERGY_RULES:
        if ability1 == own_ability:
            for ab, bonus in ability_bonus_map.items():
                if ability2 == ab:
                    ability_score += -1 * bonus # Get negative synergy

            for t in type2:
                if t in type_bonus_map:
                    ability_score += -1 * type_bonus_map[t] # Get negative synergy

    # Get bonuses for certain abilities
    if ability1 == "Intimidate" and p2.base_stats['atk'] > 100:
        ability_score += 1.0
    elif ability1 == "Armor Tail":
        priority = False
        for move in moves2:
            if hasattr(move, 'priority') and move.priority and move.priority > 0:
                priority = True

        if priority:
            ability_score += 1

    return ability_score
```

Figure 3.6 Matchup Calculations

Source: https://github.com/nicholaswisee/Makalah-Matdis

F. Framework Constructions

The core of this framework lies in the 2 graphs mentioned before, which are the matchup score graph and the team synergy graph. The first of which to be constructed is the matchup score graph. To build this bipartite graph, each the nodes are separated into two groups, which are the player's team nodes and the opponent's team nodes. After doing so, each node in the player's team will be made adjacent to every node in the opponent's team. The weight of the edge will be calculated using the sum of the effectiveness, resistance, and the ability scores. The formation of the graph is as follows.

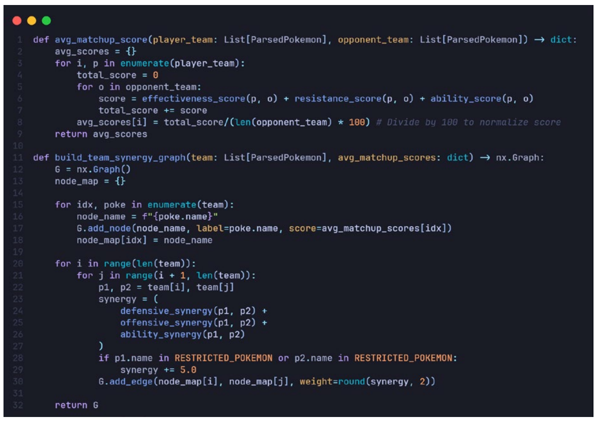


```python
def build_matchup_graph(player_team: List[ParsedPokemon], opponent_team: List[ParsedPokemon]) -> nx.Graph:
    G = nx.Graph()

    player_nodes = [f"{p.name}#P{i}" for i, p in enumerate(player_team)]
    opponent_nodes = [f"{o.name}#O{j}" for j, o in enumerate(opponent_team)]

    for name, p in zip(player_nodes, player_team):
        G.add_node(name, bipartite=0, label=p.name)

    for name, o in zip(opponent_nodes, opponent_team):
        G.add_node(name, bipartite=1, label=o.name)

    for i, p in enumerate(player_team):
        for j, o in enumerate(opponent_team):
            score = (
                effectiveness_score(p, o) +
                resistance_score(p, o) +
                ability_score(p, o)
            )
            G.add_edge(player_nodes[i], opponent_nodes[j], weight=round(score, 2))

    return G
```

Figure 3.7 Formation of The Matchup Graph

Source: https://github.com/nicholaswisee/Makalah-Matdis

The next step is to form the team synergy graph, which consists of a K_6 graph. Each edge weight is calculated with the formula previously mentioned, which is the total sum of the offensive, defensive, and ability synergy. However, in this graph, each node also has their own weight, which will be calculated as the average matchup score based on the previous graph, the calculations of which are as follows.

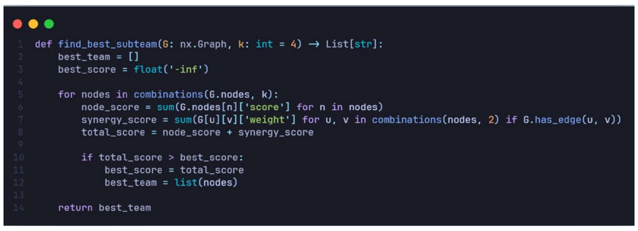


```python
def avg_matchup_score(player_team: List[ParsedPokemon], opponent_team: List[ParsedPokemon]) -> dict:
    avg_scores = {}
    for i, p in enumerate(player_team):
        total_score = 0
        for o in opponent_team:
            score = effectiveness_score(p, o) + resistance_score(p, o) + ability_score(p, o)
            total_score += score
        avg_scores[i] = total_score/(len(opponent_team) * 100) # Divide by 100 to normalize score
    return avg_scores

def build_team_synergy_graph(team: List[ParsedPokemon], avg_matchup_scores: dict) -> nx.Graph:
    G = nx.Graph()
    node_map = {}

    for idx, poke in enumerate(team):
        node_name = f"{poke.name}"
        G.add_node(node_name, label=poke.name, score=avg_matchup_scores[idx])
        node_map[idx] = node_name

    for i in range(len(team)):
        for j in range(i + 1, len(team)):
            p1, p2 = team[i], team[j]
            synergy = (
                defensive_synergy(p1, p2) +
                offensive_synergy(p1, p2) +
                ability_synergy(p1, p2)
            )
            if p1.name in RESTRICTED_POKEMON or p2.name in RESTRICTED_POKEMON:
                synergy += 5.0
            G.add_edge(node_map[i], node_map[j], weight=round(synergy, 2))

    return G
```

Figure 3.8 Team Synergy Graph Formation

Source: https://github.com/nicholaswisee/Makalah-Matdis

To get the final result of four Pokémon, a calculation is done to retrieve the subgraph of four Pokémon with the highest weight. The heaviest weight means that the total sum of the edge values and node values are the highest among combination. To calculate such values, a brute-force approach is used.



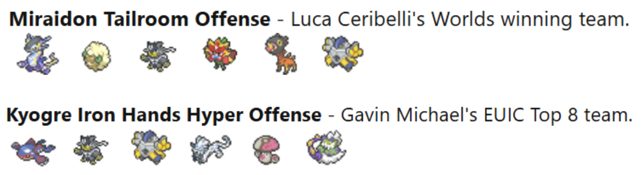
```python
def find_best_subteam(G: nx.Graph, k: int = 4) -> List[str]:
    best_team = []
    best_score = float('-inf')

    for nodes in combinations(G.nodes, k):
        node_score = sum(G.nodes[n]['score'] for n in nodes)
        synergy_score = sum(G[u][v]['weight'] for u, v in combinations(nodes, 2) if G.has_edge(u, v))
        total_score = node_score + synergy_score

        if total_score > best_score:
            best_score = total_score
            best_team = list(nodes)

    return best_team
```

Figure 3.9 Heaviest Subgraph Calculation

Source: https://github.com/nicholaswisee/Makalah-Matdis

IV. Usage

This framework was built to be able to analyze most teams that are legal for Regulation G of Pokémon VGC. To effectively analyze the results of this framework, two high quality standard teams are going to be used and compared: the World Championship winning team (Miraidon Tailroom Offense) made by Luca Ceribelli as the player team and a top 8 team in the EUIC tournament (Kyogre Iron Hands Hyper Offense) made by Gavin Michael as the opponent team.


Miraidon Tailroom Offense - Luca Ceribelli's Worlds winning team.

Kyogre Iron Hands Hyper Offense - Gavin Michael's EUIC Top 8 team.

Figure 4.1 Sample Teams for Framework Testing
Source: https://www.smogon.com/forums/threads/vgc-25-regulation-g-sample-teams-thread.3747193/

After the team has been parsed, the players' team will be analyzed against the opponent's team using the framework. The results are given as below.

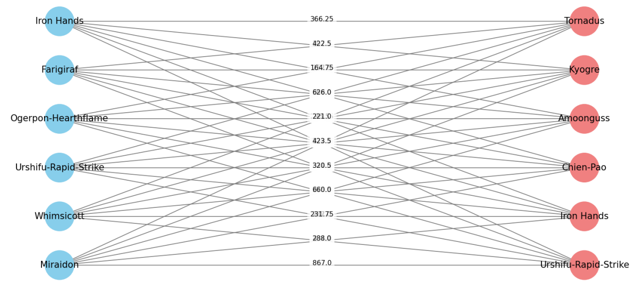


Figure 4.2 Matchup Bipartite Graph
Source: https://github.com/nicholaswisee/Makalah-Matdis

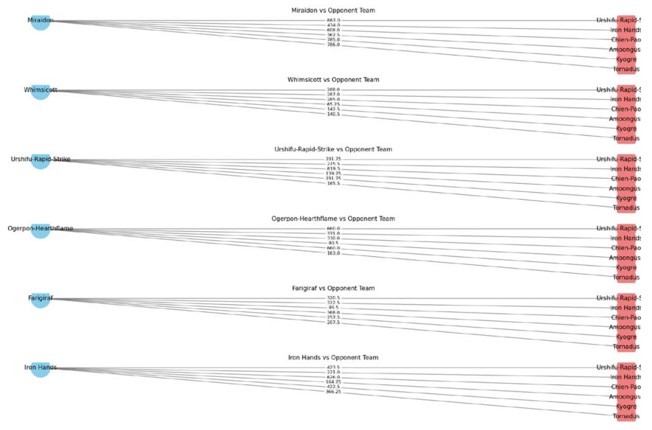

Miraidon vs Opponent Team
Whimsicott vs Opponent Team
Urshifu-Rapid-Strike vs Opponent Team
Ogerpon-Hearthflame vs Opponent Team
Farigiraf vs Opponent Team
Iron Hands vs Opponent Team

Figure 4.3 Matchup Bipartite Graph (Each Pokémon Separated)
Source: https://github.com/nicholaswisee/Makalah-Matdis

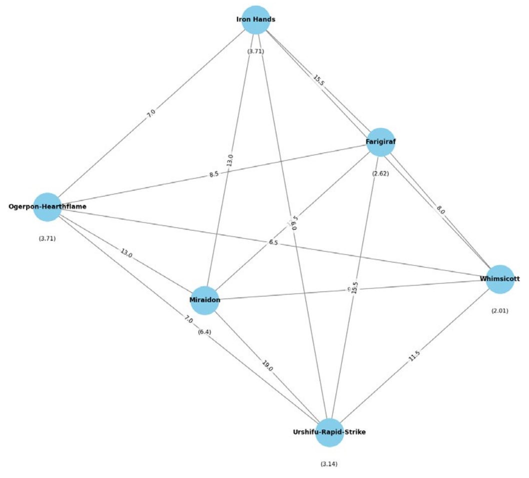


Figure 4.4 Team Synergy Graph
Source: https://github.com/nicholaswisee/Makalah-Matdis

From the figure above, it is seen that the graph scores are consistent with Pokémon type and ability relationships. The heaviest subgraph is determined in the following figure.

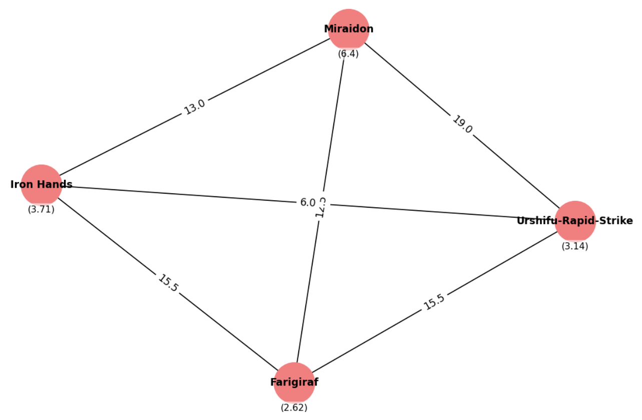


Figure 4.5 Heaviest Subgraph of 4 Nodes
Source: https://github.com/nicholaswisee/Makalah-Matdis

These results are consistent when we match these individual Pokémon against the enemy Pokémon in terms of types. The enemy consists of their strongest core being water types and the subgraph chose 2 Pokémon that are very strong against those water types, which are Iron Hands and Miraidon, being Electric types. Farigiraf having Armor Tail as an ability also helps against the enemy team, as they have 3 Pokémon that use priority moves.

V. Conclusion

This framework provides an application of understanding the core mechanics of competitive Pokémon doubles. By quantifying the relationships between each Pokémon and both it's teammates and opponents, it provides a much more proper and logically sound approach rather than solely relying on instincts and feelings. The main weakness of this framework is mainly caused by the fact that Pokémon teams don't always follow the same structure and certain Pokémon may be able to fill certain roles it usually doesn't. Either way, this analysis helps us to understand another interesting application of graph theory, which is to model the relationships of the members of a team and even against another team.

VI. Appendix

The GitHub repository hosting the source code for this project may be accessed here: GitHub Repository

VII. Acknowledgement

The author would like to thank God Almighty for his everlasting care and for providing the strength and will to be able to complete this paper. The author would also like to thank the lecturer of the IF1220 Discrete Mathematics course, Dr. Ir. Rinaldi Munir, M.T. for his invaluable guidance, insightful lectures, and continuous support in throughout the semester. The author also wishes to acknowledge the role of Wolfe Glick as an insightful figure in the competitive Pokémon community, whose content has served as both inspiration and reference for many parts of this paper. Finally, the author would also wish the utmost gratitude towards their family and their everlasting support.

References

[1] R. Munir, "Graf (Bagian 1)". Informatika STEI ITB, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf (accessed: 18 Jun. 2025).

[2] Wolfe Glick, "Introduction to Competitive Pokemon", VGC Guide, 2025. [Online]. Available: https://www.vgcguide.com/introduction-to-competitive-pokemon (accessed: 19 Jun. 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, 20 Juni 2025

[illegible]

Nicholas Wise Saragih Sumbayak

13524037