Team Counter-Selection Games

Dawson Crane, Zachary Holmes, Taylor Tadziu Kosiara, Michael Nickels, and Matthew Spradling
Dept. of Computer Science, Engineering, and Physics
University of Michigan - Flint
Email: dawsoncr@umich.edu, zaholmes@umich.edu, tkosiara@umich.edu, minickel@umich.edu, and mjspra@umich.edu

***Abstract*—We model team-versus-team contests with limited team size and an open pool of team member candidates. In this setting, candidates with a higher win rate against the open pool may be considered the "meta." Simply selecting the meta candidates leaves the team open to be countered by off-meta candidates which have lower overall win rates but high win rates against the meta in particular. A central authority in this model selects team members in hopes to counter the team composition they believe will be selected by an opponent. We present algorithms that generate a team of candidates based on observed metas and given that both parties have knowledge of pairwise election battle wins of the usable candidate pool. We provide different methodology to generate teams and analyze the teams generated by our algorithms using Pokémon GO team compositions to test them.**

***Index Terms*—Competitive Game Theory, Algorithm Design, Candidate Selection, Team Formation Games, Epistemic Reasoning, Pokémon GO**

I. Introduction

Throughout different genres of video games, there lies competitive situations where there is a need to select candidates that will come out victorious. Situations like this occur in games such as League of Legends, where the player must choose a champion to face off against other champions. It could also occur in a political setting where a candidate must be chosen to run in a primary election against a pool of candidates. Even choosing players to pick in a pickup game of baseball or volleyball could be considered as an instance of this situation. When presented with a plethora of candidates to choose from, it can be difficult to decide, especially when there are so many factors involved. One must consider how their candidate performs against other candidates, and must think about which candidates their opponent will choose.

When choosing candidates, most players are not choosing blindly. Many draw upon their knowledge of previous matches or on their knowledge of other players' preferences in order to help decide which candidates to choose [1]. Players can also share their knowledge of the game online and create websites that can showcase various strategies or ranking of candidates. This shared knowledge of a game can ultimately affect how someone plays it [2], [3]. This, in turn, will lead to the formation of *metas* which define popular competitive strategies. With this knowledge players can make more informed decisions and refine their approach to team building.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

A meta refers to outside knowledge of the game. This can include strategy and knowledge of how the candidates rank against each other. Metagaming, on the other hand, is when a player uses this outside knowledge to their advantage to make crucial gameplay decisions [4]. In League of Legends, certain champions will become the meta picks because they perform better than the majority of the other champions. In other words, they will be picked more because of their better performance. However, just because certain candidates are in a meta does not mean they are necessarily the best set of candidates to choose. There can be other meta strategies that have higher win rates than what is in the current meta [5]. After some time, champions that beat the established meta will start to become more popular until they ultimately establish an new meta. That means that the metas will rotate as each new meta eventually gets replaced by a counter meta [6], [7].

We propose to utilize our knowledge of various metas, observe the patterns of how they shift from one to the next, and create balanced team compositions that give a greater probability of winning. We refer to this problem model as a *Team Counter-Selection Game*.

In Section II, we define the Team Counter-Selection Game (TCSG) model. Section III outlines related work in team formation games, epistemic reasoning, and competitive game theory. Sections IV and V define exact algorithms for maximizing either the coverage of a team or its win rate. Section VI details a heuristic probabilistic approach for cases where information is broadly known and choices need to be less predictable. Sections VII details experimental results with our algorithms to generate teams in the mobile game Pokémon GO. Finally, we outline future work and conclusions in Sections VIII and IX.

II. Team Counter-Selection Games

An instance of Team Counter-Selection Game (TCSG) consists of:

- A: a set of candidates that Player 1 can choose from
- B: a set of candidates that Player 2 (the opponent) can choose from
- M: a function over $A \times B$. For any a in A and $b \in B$: M[a][b] is 1 when a defeats b and 0 when otherwise.
- n: the team size to be generated

A solution to an instance of TCSG is $C \subseteq A$ s.t. $|C| = n$.

In a TCSG, both players will select teams in hopes to defeat one another in pairwise competitions between candidates. For optimization problems in this setting, we consider the

perspective of Player 1 choosing a team to defeat Player 2. A legal solution for Player 1 is a team of candidates $C \subseteq A$. An optimal solution maximizes the probability that C defeats the team $D \subseteq B$ to be selected by Player 2.

An exact algorithm for this problem is useful when you have more information than your opponent. For instance, if you know exactly which candidates your opponent will choose, or their subset is significantly smaller than the open field, an exact algorithm can determine the best counters for that smaller subset. This is valuable if you anticipate your opponent will choose their team from only a limited meta. We define such solutions in Sections IV and V.

When every party participating in a competition has the same information, and teams are not known in advance, agents capable epistemic reasoning (reasoning about knowledge) may attempt to choose candidates to counter likely choices of an opponent. Consider if everyone used an exact algorithm to do so. In this case, all parties would get the same best team. When everyone gets the same best team it makes it easy to predict these teams and counter them. But then, knowing that they know that we know, we might choose to counter their counter. How many levels deep should this counter-picking go? A probabilistic approach is designed to work against counter-picking by not necessarily guaranteeing the choices made. Candidates are weighted so that you can get a good team while not being so easily predictable. We define such a solution in Section VI.

III. Related Work

Epistemic reasoning is a standard concept in competitive game theory [8]. It has been studied in the context of teamwork in multi-agent systems [9] and in modeling reasoning in competitive play for general Bayesian games with limited knowledge [10]. A broadly generalized model for multi-agent epistemic planning considers the team formation problem in terms of agents, atomic fluents, actions, states, and goal conditions [11]. To our knowledge, epistemic reasoning has not yet been applied to our simplified competitive team formation model. In our model, the goal condition is always to defeat the opponent's unknown team based upon the known results of pairwise contests between candidates. Epistemic reasoning in our setting is from the perspective of the player forming the team rather than that of the team member candidates.

In "Multiwinner Elections With Diversity Constraints", the authors looked at the computational complexity of choosing a committee which meets certain constrains such as performance requirements or gender diversity [12]. We similarly select candidates in a form of multi-winner election, but our constraints are simplified to maximizing wins rates in pairwise contests against opposing teams.

There is also a notion proposed of the Condorcet winner, which is defined as a candidate who wins against all of their pairwise elections. However, in most applicable cases, there won't be a Condorcet winner. Elkind, Lang, and Saffidine define a Condorcet winning set which is a collection of candidates who dominate a majority of the other candidates [13]. Similarly, if a Condorcet winner exists in a TCSG instance, then this candidate should be chosen for any team.

Our work is structured similarly to "Team Contests With Multiple Pairwise Battles". In this work, the authors propose a model for team contests where two equal-size teams of candidates are matched against each other. Each candidate on one team is matched to an opponent on the other team in a component battle. The results of each component battle are independent from each other. In other words, a team member winning or losing in one component battle won't have any effect on future component battles. The winning team is decided by which side wins the most component battles [14]. The work in our paper is modeled in a similar nature to this, where we compare candidates in separate independent component battles. However, instead of having a candidate matched to only one candidate on the opposing team, we observe how a candidate compares to every other candidate on the opposite team.

When looking at candidates to assemble a team, it is important to know how they perform against each other. One way is to gather match data like in the previously mentioned studies. A different way to simulate a match between two candidates. Montes proposes a battle simulation for Pokémon. This simulation takes two Pokémon as input that will battle each other in a one on one match. The simulation will make a decision tree that will consider all legal moves the Pokémon can make. It will then analyze each node in order to determine the Pokémon's best action [15]. Using this can help you determine which Pokémon will win the one on one fight. In our testing that we describe later in the paper, we use a different battle simulator provided by the website PvPoke in order to collect the data for our tests [16].

Team building algorithms have been considered for a variety of games. "How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2" uses a machine learning algorithm to determine the best candidates based on a given enemy team. By taking in both the team compositions and the results of over 50,000 top rated matches, Perry and Conley's greedy search algorithm outputs the list of available candidates sorted by descending probability of victory [17]. While Perry and Conley optimize individual heroes against an individual composition, we seek to optimize a composition against an entire field of possible compositions the player may encounter.

In "Team Recommendation for the Pokémon GO Game Using Optimization Approaches," the authors test three different optimization algorithms that select an ideal team when given an opposing team [18]. Our model differs in that the opposing team is not known in advance.

"A Time-Efficient Competitive Pokémon Team-Building Algorithm" focuses on building an optimal team in Pokémon given user input of a Pokémon they are interested in using on their team, and then builds the team around that Pokémon. By using typing, base stat total, and how popular a candidate is in the meta, a candidate scoring system is created that is unique to the main series of Pokémon games. This work optimizes around the a priori preferences of the player [19].

Our algorithms instead uncover new candidates and team compositions the player may not have otherwise considered.

IV. Maximizing Coverage

Maximizing coverage means to select $C \subseteq A$ that defeats the greatest number of different candidates in B. The rationale is to increase the probability you will be faced with a favorable match-up. The MaximizeCoverage greedy local search algorithm (see Algorithm 1) iteratively selects $a \in A$ maximizing the number of unbeaten candidates in B that a defeats.

TABLE I
TCSG INSTANCE WHERE A = [U,V,W], B = [V,W,X,Y,Z] AND M IS DEFINED BY THE BOOLEAN MATRIX WHERE $M[i][j] = 1$ WHEN i DEFEATS j AND $M[i][j] = 0$ OTHERWISE.

	V	W	X	Y	Z
U	1	1	0	1	0
V	0	0	1	1	0
W	0	0	0	1	1

TABLE II
MODIFIED TCSG INSTANCE FOLLOWING FROM TABLE I AFTER FIRST ITERATION OF MAXIMIZECOVERAGE HAS SELECTED CANDIDATE U.

	V	W	X	Y	Z
U	0	0	0	0	0
V	0	0	1	0	0
W	0	0	0	0	1

Algorithm 1 MaximizeCoverage (matrix A, matrix B, matrix M, float c)

```
Team[]
M[0] = M
for all i team candidates do
  BestCandidateValue
  for all a candidates ∈ A do
    bCandidateSum
    for all b candidates ∈ B do
      bCandidateSum += M[a][b]
    end for
    if bCandidateSum > BestCandidateValue then
      BestCandidateValue = bCandidateSum
      Team[i] = b
    end if
  end for
  M[i] = M[i − 1]
  for all a candidates ∈ A do
    for all b candidates ∈ B do
      if M[i][BestCandidate][b] == 1 then
        M[i][a][b] = 0
      end if
    end for
  end for
end for
```

MaximizeCoverage runs in time $O(n|A||B|)$. Calculating the total wins for a single candidate requires $O(|A||B|)$ operations to sum total wins for each $a \in A$ over B and then update the battle matrix. This update removes the utility for beating candidates covered by the newest team member (see Tables I and II). This process is performed once for each of n selected candidates.

V. Maximizing Team Wins

Maximizing team wins means to generate a team that maximizes the number of team compositions it defeats. We assume that candidates in a team will face enemy candidates one on one in independent matches and that the results of one match have no impact on the next. The rationale is to increase the number of compositions the chosen team can defeat.

Let $C \subseteq A$ and $D \subseteq B$ be two teams. We say that the *battle score* of A against B is equal to the total number of wins candidates in A achieve against candidates in B. For example, if $A = i, j$, $B = x, y$, i defeats x and y, and j defeats y, then the battle score of A against B is equal to 3. We say that A defeats B when the battle score of A is greater than the battle score of B.

CheckIfTeamWins (see Algorithm 2) computes the battles scores for two teams G and H given the battle matrix M. It returns true if G defeats H and false otherwise, with ties treated as a loss for G. The time complexity of CheckIfTeamWins is $\mathcal{O}(n^2)$ as $|G| = |H| = n$.

MaximizeTeamWins (see Algorithm 3) compares each team composition $G \subseteq A$ to each composition $H \subseteq B$. Each iteration, a *winCounter* is accumulated for G equal to the total number of compositions from B that G defeats. If G has a greater *winCounter* than the previous best, G is stored as the new best team. We continue this process until we have checked every possible team composition $G \subseteq A$. The algorithm then returns the team maximizing *winCounter*.

Algorithm 2 CheckIfTeamWins (Team $\vec{G}$, Team $\vec{H}$, BattleMatrix[])

```
battleScoreG ← 0
battleScoreH ← 0
for all g candidates ∈ G⃗ do
  for all h candidates ∈ H⃗ do
    if M[g][h] == 1 then
      battleScoreG++
    else
      battleScoreH++
    end if
  end for
end for
if battleScoreG > battleScoreH then
  return True
else
  return False
end if
```

Algorithm 3 MaximizeTeamWins (Set $\vec{A}$, Set $\vec{B}$, matrix M)

```
bestTeam = null
bestTeamWins ← 0
for all G candidates ∈ A do
  winCounter ← 0
  for all H ∈ B do
    if CheckTeamWins(G,H,M) == 1 then
      winCounter++
    end if
  end for
  if winCounter >bestTeamWins then
    bestTeamWins ← winCounter
    bestTeam ← G
  end if
end for
return bestTeam
```

When going through MaximizeTeamWins, the number of team compositions in a meta will equal

$$C(n,r) = \frac{P!}{n!(P-n)!}$$

where P is the population of candidates in a meta and n is the size of the team. Because of this, the time complexity for finding every team composition will be

$$O\frac{P^n}{n!}$$

In other words, finding all team compositions for metas A and B will be

$$O\frac{|A|^n}{n!}$$

and

$$O\frac{|B|^n}{n!}$$

respectively. With this, we can observe that the overall time complexity of MaximizeTeamWins is

$$O\frac{|A|^n|B|^n n^2}{n!^2}$$

where $|A| \geq n$ and $|B| \geq n$. When the team size n is some constant c, the time complexity of MaximizeTeamWins simplifies to the polynomial

$$O|A|^c|B|^c$$

VI. Weighted Probability Team Generator

By giving weighted probabilities to candidates, we generate teams which may sacrifice higher win rates in exchange for being more difficult to predict. This strategy is valuable when both the player and the opponent may employ epistemic reasoning in attempting to counter-select candidates for their teams. We expect this process to have worse performance than MaximizeTeamWins (Algorithm 3) in terms of maximizing the % of the meta covered but to have greater success in practice when faced with counter-selection.

WeightedProbabilityTeamGenerator starts out by generating the weights of all the candidates in meta A by using the GenerateWeights function. GenerateWeights has a time complexity of $O(|A||B|)$ as it must sum the weights of each candidate in A by comparing them against each candidate in B.

In Table III, we can see that U's value is 3, while V and W's values are 2, so the weighted probability for candidate U would be $3/(3+2+2)$. Once we find the weight values for every candidate in A, we then randomly choose the candidate based on this weighted probability. Next, similarly to how we handled the Maximum Coverage Algorithm, we modify M to remove the influences of previous team member in C. With a new modified version of M in Table IV, we continue the process starting from generating the weights of every candidate in B based on the new modified version of $M[i]$.

TABLE III
TCSG instance following from Table I with total values and weights computed.

	V	W	X	Y	Z	Total	Weight
U	1	1	0	1	0	3	0.43
V	0	0	1	1	0	2	0.28
W	0	0	0	1	1	2	0.28

TABLE IV
TCSG instance following from Table II with total values and weights computed.

	V	W	X	Y	Z	Total	Weight
U	0	0	0	0	0	0	0
V	0	0	1	0	0	1	0.5
W	0	0	0	0	1	1	0.5

Algorithm 4 GenerateWeights (Set $\vec{A}$, Set $\vec{B}$, BattleMatrix[])

```
Weights[]
TotalValue = 0
for all a candidates ∈ A⃗ do
  CandidateXSum = 0
  for all b candidates ∈ B⃗ do
    CandidateXSum += BattleMatrix[a,b]
  end for
  Weights[x] = CandidateXSum
  TotalValue += CandidateXSum
end for
for all a candidates ∈ A⃗ do
  Weights[a] = Weights[a]/TotalValue
end for
return Weights[a]
```

After GenerateWeights returns, the WeightedProbablyTeamGenerator randomly selects a candidate from A with probability based upon the candidate's computed weight. This random selection is performed in $O(1)$. Next we modify the battle matrix to remove the influences of the candidate we have chosen. This is done in $O(|A||B|)$. This process repeats for n iterations, selecting one new team member each iteration. The overall time complexity of WeightedProbabilityTeamGenerator is $O(n|A||B|)$.

Algorithm 5 WeightedProbabilityTeamGenerator(A, B, M, n)

```
for all Candidates, i in Team do
    M[0] = M
    Weights = GenerateWeights(A, B, M[i − 1])
    Team[i] = RandomlyChooseCandidate(A, Weights)
    M[i] = M[i − 1]
    for all Candidate, a, in A do
        for all Candidate, b, in B do
            if M[i][Team[i]][b] == 1 then
                M[i][a][b] = 0
            end if
        end for
    end for
end for
```

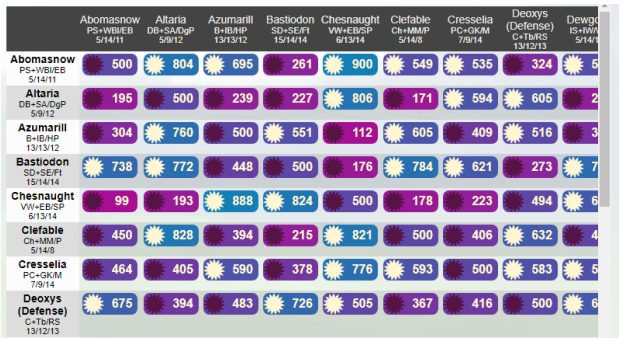

Fig. 1. Example of Battle Matrix using the Great League Meta candidate pool on PvPoke.com

VII. Testing and Results

A. Experimental Methodology

We tested the efficacy of our algorithms by simulating team selection in Pokémon Go player versus player battles. Pokémon Go is a popular mobile game where players collect, trade, and battle with different species of avatars called Pokémon [20]. A Pokémon GO battle consists of two players each battling with a team of 3 Pokémon taken from their individual collections. The exact membership of an opponent's collection is unknown to the player, and battle teams are not revealed until the battle begins. It is known in advance which species of Pokémon should win in pairwise match ups given perfect play and expected moves. Therefore, a player can choose a battle team based upon epistemic reasoning on which team the opponent is likely to use.

In our experiments, P is the set of all Pokémon species available in the game. Since 1 versus 1 match-ups are predetermined by the game data, we can obtain a square matrix M of pairwise match ups for candidates in P. We implemented Meta Generator in Python 3 using an Excel sheet as input. Our data for M comes from the website PvPoke.com [16]. PvPoke is an open source website that is used to rank Pokémon and simulate player versus player match-ups directly from the Pokémon Go GameMaster file. By going to the battle section of the website, you can choose to do a matrix battle, and you can select Pokémon to test. It is possible to export this data from the website into a spreadsheet. It outputs a table similar to our model that shows how every Pokémon performs against every other Pokémon in the matrix. Figure 1 below shows an example of a matrix battle on PvPoke.com. Every Pokémon's performance in a battle is listed in the matrix as an integer called a battle rating. If this number is above 500, that means the Pokémon has won.

We tested several different groups of candidates using WeightedProbabilityTeamGenerator and MaximizeTeamWins. Each of our tests pulled Pokémon from one of the three leagues, or pools of candidates, found in the game. These are the Great League, Ultra League, and Master League. The league you participate in determines which subset of Pokémon in P you can use. Each Pokémon has a calculated number called CP, which is an integer that rates how strong that Pokémon is in battle. Pokémon with 1500 CP or less can participate in Great League, Pokémon with 2500 CP or less can participate in Ultra League, and in Master League there is no cap to the CP. Some of these leagues also have special rule sets that limit the kinds of Pokémon you can use on your team, called a cup. For example, in the premier cup, legendary and mythical Pokémon are banned from participating, which allows the creation of a different meta than the "open" leagues. In addition to pairwise matchup data, PvPoke offers ratings of Pokémon in the various leagues and cups based upon their relative win rates and the frequency which which they are used in practice. These ratings give us a baseline for comparing the results of our algorithms.

Seven runs of the MaximizeTeamWins algorithm for teams of size two were tested using subsets of the Great League meta. The Great League meta consists of the top 48 candidates from GO Battle Log. GO Battle Log ranks Pokémon based upon user submitted usage data [21]. We generated each subset using a procedure called IterativeMetaGenerator (see Algorithm 6) which determines the set of candidates defeating at least c percent of the previous iteration. We took $c = 0.5$.

Given a set of candidates, we refer to the subset of candidates which are currently being utilized by the population of players as the current "meta." The IterativeMetaGenerator simulates "waves" of changes in the meta as players attempt to counter recently seen teams.

B. Iterative Meta Example

Consider a modified version of the *Rock, Paper, Scissors* game. In this variation, a fourth object called "Rock++" is included. Rock beats Scissors, Paper beats Rock, Scissors beats Paper, and Rock++ beats both Rock and Scissors. In this example, we let $c = 50\%$ and $D[0]$ = {Rock, Paper, Scissors, Rock++}. In the first iteration, the algorithm generates a meta $D[1]$ = {Rock++, Paper} countering $D[0]$ as seen in Table V.

The next iteration in Table VI considers the value of the entire field against the previous meta, $D[1]$. Because we only expect the meta Paper and Rock++ to be played, only these two objects have value to be defeated. As a result, Rock++

Algorithm 6 Iterative Meta Generator (int $|P|$, matrix M, float c)

D = [1,1,1,1,....,$|P|$-1] {Matrix of values of D, where candidates in D meet the minimum percentage of c from previous iteration}
E = [[]] {Where each array is the percentages that a candidates defeats the previous candidate}
I = 0 {Previous Iteration}
while $D[I]! = D[i]$ from i = 0 to $i <$ **do**
  int TotalCandidatesOfPreviousIteration = 0;
  for all x candidates $\in P$ **do**
    TotalCandidatesOfPreviousIteration += D[I][x]
  end for
  for all x candidates $\in P$ **do**
    sum = 0
    for all y candidates $\in P$ **do**
      sum += $M[x][y] * D[I][y]$;
    end for
    if (sum/TotalCandidatesOfPreviousIteration) $> c$ **then**
      set value of candidate to 1 in D[I+1] then set candidate's percentage in E[I+1]
    end if
  end for
end while
return D and E

becomes obsolete and {Paper, Scissors} becomes the meta D[2].

TABLE V
META GENERATED IN ITERATION 1 INCLUDES PAPER AND ROCK++.

	Rock	Paper	Scissors	Rock++	Sum	%	D
Rock	0	0	1	0	1	25	0
Paper	1	0	0	1	2	50	1
Scissors	0	1	0	0	1	25	0
Rock++	1	0	1	0	2	50	1

TABLE VI
META GENERATED IN ITERATION 2 INCLUDES PAPER AND SCISSORS.

	Paper	Rock++	Sum	%	D
Rock	0	0	0	0	0
Paper	0	1	1	50	1
Scissors	1	0	1	50	1
Rock++	0	0	0	0	0

TABLE VII
META GENERATED IN ITERATION 3 INCLUDES ROCK, SCISSORS, AND ROCK++

	Paper	Scissors	Sum	%	D
Rock	0	1	1	50	1
Paper	0	0	0	0	0
Scissors	1	0	1	50	1
Rock++	0	1	1	50	1

There are a finite number of combinations of D[I]. Reaching a duplicate meta forms a closed loop. In this example the 4th iteration duplicates the 1st iteration, forming a loop as seen in Table VIII. This gives us a set of 4 metas to consider; the initial meta plus the metas formed in iterations 1, 2 and 3.

TABLE VIII
METAS GENERATED IN ITERATIONS $[0, 1, 2, 3, 4]$. BECAUSE 1 AND 4 ARE IDENTICAL METAS, THIS FORMS A LOOP OVER ITERATIONS 1,2 AND 3.

	0	1	2	3	4
Rock	1	0	0	1	0
Paper	1	1	1	0	1
Scissors	1	0	1	1	0
Rock++	1	1	0	1	1

C. Experimental Results

Tables IX, X, XI, XII describe the results of our experiments. In each table, each row represents a trial attempting to counter a single meta. *% P* is the percentage of Pokémon from the population which are found in the meta we are attempting to counter in that trial. *% T* is the percentage *threats* (according to PvPoke battle simulations) not found in that meta. *% M* is the percentage of Pokémon from that meta which our selected team has *coverage* for; meaning that at least one Pokémon on our selected team can defeat that percentage of Pokémon from the meta. Finally, we simulated battles between our selected team and each possible team from the current meta. *% W* is the percentage of all such teams that our selected team defeats.

TABLE IX
MAXIMIZETEAMWINS DUOS GREAT LEAGUE META (48 POKÉMON)

Trial #	% P	% T	% M	% W
1	20.83%	100.00%	90%	77.78%
2	37.50%	85.70%	94.44%	84.97%
3	43.75%	91.67%	80.95%	73.81%
4	47.92%	80.00%	91.30%	69.17%
5	50.00%	69.20%	90.48%	75.72%
6	56.25%	8.30%	92.59%	73.50%
7	58.33%	90.00%	96.43%	71.96%

For the Great League Meta with teams of size 2, Table IX, MaximizeTeamWins averaged 90.88% coverage over seven trials with an average win rate of 75.27%. We noted one outlying case in trial 6 where the team chosen in this case (Shadow Machamp and Swampert) happens to have many potential threats, which prevent it from performing well in practice. Both have lower defense, higher attack power and neither could survive long enough to reach its damage potential without a more defensive Pokémon to back them up.

In the Master League Premier Cup Meta, Table X, we tested teams of size 3 for a meta of 26 candidates. The average coverage was 100.00% over six trials with an average win rate of 97.33%. This was a surprising increase in the quality of performance when the team size increased from 2 to 3. We believe that the brute force algorithm was better able to exploit

weaknesses in the meta with this additional team member and that perhaps only 2 Pokémon would have trouble covering each others weaknesses entirely.

TABLE X
MAXIMIZETEAMWINS TRIOS MASTER LEAGUE PREMIER CUP (26 POKÉMON)

Trial #	% P	% T	% M	% W
1	15.38%	100.00%	100.00%	100.00%
2	23.08%	100.00%	100.00%	100.00%
3	30.77%	90.00%	100.00%	91.07%
4	42.31%	88.89%	100.00%	98.79%
5	46.15%	77.78%	100.00%	94.09%
6	53.85%	57.14%	100.00%	100.00%

In the Great League Kanto Cup and Master League Premier cup, we tested the efficacy of the WeightedProbabilityTeamGenerator algorithm.

For the Great League Kanto Cup meta, Table XI, we tested teams of size 2 for a meta of 31 candidates. The average coverage was 77.04% over 9 trials with an average win rate of 86.62%. We noted one outlying case in trial 4 where % coverage decreased dramatically. The team chosen in this case (Shadow Victreebel and Alolan Ninetails) suffers a similar fate to Shadow Machamp and Swampert in experiment 1. Both duos are low defense, high attack, and are unable to amass their potential together.

For the Master League Premier Cup meta, Table XII, we tested teams of size 3 for a meta of 26 candidates. The average coverage was 96.03% over 6 trials with an average win rate of 97.85%. One outlying case in trial 5 comes from a team consisting of Gengar, Electivire, and Snorlax. This team tends to be weak against Ground-type Pokémon, which are more prevalent in this meta. We still found significant improvement with team size 3 compared to team size 2 with the second algorithm.

TABLE XI
WEIGHTEDPROBABILITYTEAMGENERATOR DUOS KANTO CUP (48 POKÉMON)

Trial #	% P	% T	% M	% W
1	29.03%	71.43%	66.67%	91.67%
2	29.03%	80.00%	88.89%	72.22%
3	29.03%	80.00%	77.78%	97.22%
4	41.94%	16.67%	53.84%	96.15%
5	41.94%	40.00%	69.23%	87.18%
6	41.94%	62.50%	76.92%	64.10%
7	32.26%	50.00%	80.00%	86.67%
8	32.26%	100.00%	100.00%	86.67%
9	32.26%	71.43%	80.00%	97.78%

TABLE XII
WEIGHTEDPROBABILITYTEAMGENERATOR TRIOS MASTER LEAGUE PREMIER CUP (26 POKÉMON)

Trial #	% P	% T	% M	% W
1	23.08%	88.89%	100.00%	95.00%
2	23.08%	83.33%	100.00%	95.00%
3	46.15%	42.86%	83.33%	100%
4	46.15%	60.00%	100.00%	98.18%
5	53.85%	18.18%	92.86%	98.90%
6	53.85%	44.44%	100.00%	100%

In experiment 1 using the MaximizeTeamWins algorithm for duos in the Great League meta, Azumarill was the most frequently selected team member. On these tests Azumarill appeared in our duo team compositions seven out of twenty-three times, or around 30.4% of our total Great League duo tests done. According to GO Battle Log, Azumarill can be seen on over 6.3% of competitive teams while also being ranked number two overall on PvPoke's rankings [21]. Although being a popular choice does not necessarily imply being a good candidate to select, our MaximizeTeamWins duos did line up with our expectations that Azumarill was going to perform well. In this case, our algorithm's results seem to match the consensus of the players.

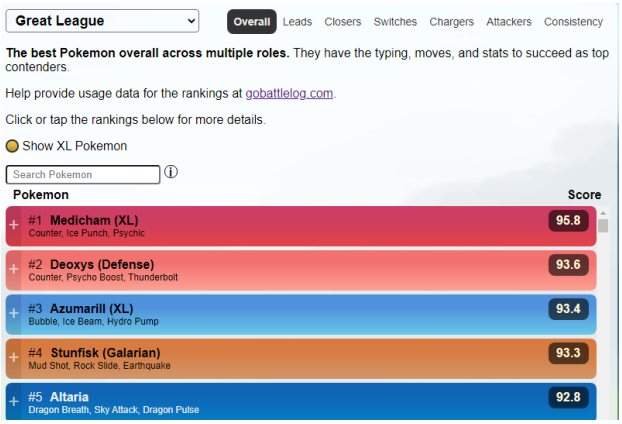

Fig. 2. The top five Pokémon in Great League according to PvPoke.com as of 4/7/21. Note that Azumarill is ranked 3rd out of 667 possible candidates.

Both algorithms were able to discover surprising candidates in the Master League Premier meta. MaximizeTeamWins discovered Shadow Gardevoir in five out of the eight total tests. With it being ranked 24th out of the 26 "meta" candidates by PvPoke, it was not expected to be chosen the most often. Shadow Gardevoir was often paired with two other candidates, Togekiss and Gyarados. In trial 1, the team of Shadow Gardevoir, Togekiss, and Gyarados held to be the strongest team composition versus the meta, holding 207 wins out of the 220 possible team compositions and 100% coverage over the candidates we sought to cover. This same team was seen again with MaximizeTeamWins in the fourth trial, winning against 163 out of the 165 possible team compositions and also holding an 100% coverage over candidates we sought

to cover. In experiment 4, WeightedProbabilityTeamGenerator also discovers Shadow Gardevoir in two unique teams for Master League Premier cup. In trial 2, Shadow Gardevoir was paired with Dragonite and Shadow Metagross allowing each to cover the others' weaknesses. In trial 6, Shadow Gardevoir, Machamp, and Rhyperior provided two sources of protection against Gardevoir's weakness to the steel typing, allowing it to become a powerful and unexpected closer.

VIII. Future Work

We structured our model so our algorithms did not have to rely on in-game information in order for our solutions to be more applicable to many different situations. However, in Pokémon Go there are mechanics that allow the team of candidates to influence the game decision based on game mechanics of hit points (HP) and stored energy that transfer over to the next candidate when the first candidate goes down. Future work can focus upon correctly diminishing the value of a candidate across multiple match-ups based on specified game mechanics. More work can also be done to test the efficiency of the team compositions that are generated from the algorithms. Since it costs many resources to build these teams in Pokémon Go, it is difficult to test the teams made in-game. However, there is an online Pokémon Go PvP Battle Simulator that has not been released to the public yet named Project Grookey [22]. Once this is complete, teams can be made and tested at no cost instead of creating curated tests using PvPoke battle matrix data and scoring metrics. An algorithm may also be developed to use probability to determine the correct meta that a player is in using data from previous matches. This algorithm could be used in conjunction with IterativeMetaGenerator to give an accurate prediction of what meta the opponent is using. With that information, we would be able to better pick a team to counter the current meta. Future work will also consider larger metas and team sizes and expand upon the application of iterative metas.

IX. Conclusion

We looked at both heuristic and exact algorithms to generate teams of various sizes in scenarios involving multiple candidate selection. We considered the exact algorithms MaximizeCoverage and MaximizeTeamWins and the heuristic algorithm WeightedProbabilityTeamGenerator. We compared the effectiveness of MaximizeTeamWins (MTW) to WeightedProbabilityTeamGenerator (WPTG). For teams of size 2 with a pool of 48 Pokémon, MTW achieved a significantly higher % coverage of the meta, averaging 90.88% versus 77.04%. By contrast, WPTG outperformed MTW in terms of win rate on these trials, averaging 86.62% versus 75.27%. For teams of size 3 with a pool of 26 Pokémon, MTW and WPTG had very similar and much higher results. Interestingly, both algorithms appear to perform better both in terms of coverage and win rates on this data set. Similarly, the difference between their quality became smaller. We are interested to see if this remains true in other games with larger team sizes, such as MOBA games with teams of size 5. Additionally, the size of the original meta may be a factor. Additional testing on larger populations for the original meta is warranted. The WPTG algorithm has an added benefits of both being faster than brute force and of being harder to predict due to the probabilistic nature of its choices in team formation.

X. Acknowledgements

This work was funded in part by the University of Michigan-Flint Undergraduate Research Opportunity Program (UROP). We thank the UM-Flint Office of Research for their support. We also thank our anonymous reviewers for their suggestions. In particular, their suggested additions and modifications to Tables IX, X, XI and XII allowed us to discover additional value for the WPTG algorithm which we have now described.

References

[1] S. Donaldson, "Mechanics and metagame: Exploring binary expertise in league of legends," *Games and Culture*, vol. 12, no. 5, pp. 426–444, 2017.

[2] G. Zielke, "Gotta win'em all: How expert play in the online community of smogon changes pokemon," 2020.

[3] C. S. Ang, P. Zaphiris, and S. Wilson, "Computer games and socio-cultural play: An activity theoretical perspective," *Games and Culture*, vol. 5, no. 4, pp. 354–380, 2010.

[4] K. Salen, K. S. Tekinbaş, and E. Zimmerman, *Rules of play: Game design fundamentals*. MIT press, 2004.

[5] C.-S. Lee and I. Ramler, "Identifying and evaluating successful non-meta strategies in league of legends," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–6.

[6] A. M. Adams and S. I. Walker, "Real-world open-ended evolution: a league of legends adventure," *International Journal of Design & Nature and Ecodynamics*, vol. 12, no. 4, pp. 458–469, 2018.

[7] D. P. Peabody, "Detecting metagame shifts in league of legends using unsupervised learning," 2018.

[8] H. Gintis, *The Bounds of Reason: Game Theory and the Unification of the Behavioral Sciences-Revised Edition*. Princeton University Press, 2014.

[9] B. Dunin-Keplicz and R. Verbrugge, *Teamwork in multi-agent systems: A formal approach*. John Wiley & Sons, 2011, vol. 21.

[10] O. Dianat *et al.*, "Representing and reasoning about bayesian games with epistemic logic," 2014.

[11] C. Muise, F. Dignum, P. Felli, T. Miller, A. R. Pearce, and L. Sonenberg, "Towards team formation via automated planning," in *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*. Springer, 2015, pp. 282–299.

[12] R. Bredereck, P. Faliszewski, A. Igarashi, M. Lackner, and P. Skowron, "Multiwinner elections with diversity constraints," *arXiv preprint arXiv:1711.06527*, 2017.

[13] E. Elkind, J. Lang, and A. Saffidine, "Condorcet winning sets," *Social Choice and Welfare*, vol. 44, no. 3, pp. 493–517, 2015.

[14] Q. Fu, J. Lu, and Y. Pan, "Team contests with multiple pairwise battles," *American Economic Review*, vol. 105, no. 7, pp. 2120–40, 2015.

[15] G. Montes, "Algorithm for pokemon battle game using minmax," *Sistemas Inteligentes: Reportes Finales Ago-Dic 2013*, p. 131, 2013.

[16] "Open-source battle simulator, rankings &; team building for pokemon go pvp," https://pvpoke.com/, accessed: 2020-4-9.

[17] K. Conley and D. Perry, "How does he saw me? a recommendation engine for picking heroes in dota 2," *Np, nd Web*, vol. 7, 2013.

[18] S. d. S. Oliveira, G. E. Silva, A. C. Gorgônio, C. A. Barreto, A. M. Canuto, and B. M. Carvalho, "Team recommendation for the pokémon go game using optimization approaches."

[19] H. Reijm, "A time-efficient competitive pokemon team-building algorithm," 2016.

[20] "Catch pokémon in the real world with pokémon go!" https://www.pokemongo.com/en-us/, accessed: 2010-04-16.

[21] Aaron, "Your personal battle analyzer + team generator for pokemon go pvp," https://gobattlelog.com/monchecker, 2021.

[22] DeveloperKhan, "Developerkhan/pogo-web," https://github.com/DeveloperKhan/pogo-web, accessed: 2021-04-02.