Ranking A Network of Smash Ultimate Players: Progress Report

William Li, David Song

21 April 2020

1 Introduction

Super Smash Bros Ultimate is a fighting game with a crossover of characters from various Nintendo games and franchises. The game series has been around since 1999, and Ultimate is its fifth game. It is a very popular game, so there are players worldwide competing to be the best, but how can we determine who really is the best?

There already exist techniques for ranking players in the game, ranging from simple metrics, such as win/loss ratios, to more complex values, such as scores under the ELO ranking system. Win/loss ratios are easy to calculate, but can change based on the opponents faced or number of games played, instead of the player's skill. On the other hand, ELO rankings are inconvenient to calculate because the ELO scores are dependent on the order in which the games were played, and that data is not always available. We want to use a network-based ranking approach, which will both take into account the skill of each player and be calculable with only aggregate game results.

2 The dataset

2.1 Finding the data

Between January and September of 2019, a Twitter user with handle "stu2b50" compiled onto the website `https://statsmash.io` a dataset `ultimate_sets.csv` with every game from over 800 different tournaments (each with 225+ competitors on average and upwards of 4000 players). The data was taken from `https://smash.gg`, a platform on which Smash tournaments are organized, and contains an entry for every game played in these tournaments, including over $250,000$ games for over $50,000$ players.

2.2 Cleaning the data

The dataset included more information than we were interested in, so using a custom Python script `clean.py`, we first trimmed the number of fields in the data and then removed entries that were not useful to this project. We were only interested in the winner, loser, and set score (for example, $3-1$ in a best-of-five set) of each game. Also, we removed entries where one of the players was not specified, where the scores were not specified, and where the game was a forfeit.

2.3 Preparing the data

Cleaning the data with the python script produced three files: `ultimate_sets_clean.csv` contained a list of games with the set score and global player IDs of the winner and loser, `ultimate_player_ids.csv` contained a mapping between global player IDs and player usernames, and `ultimate_tournament_attendance.csv` contained a list of every tournament in the dataset and the number of attendees at each tournament. The first file would be used to construct a network and the second file would be used to interpret the results in terms of players instead of IDs. The last file was useful to provide more context about the dataset at hand.

3 Building a ranking

For this portion of the analysis, we used version 2.4 of the `networkx` Python package.

3.1 A graph representation

To construct a ranking on the set of players, we decided to represent the set of games as a directed graph, such that for every game where player i defeats player j, we have a directed edge (j, i). This allowed for the possibility of cycles, and in particular, parallel edges. For the sake of conciseness, each player was identified only by their global ID (a 6-digit string), as opposed to their username (a more general string).

Mirroring the cleaned dataset, the graph has 52039 nodes connected by 175176 edges. There are 479 weakly connected components, the largest of which has 49049 nodes, comprising over 94% of players. We speculate that any nonconnectivity is a result of the cleaning process, which removed games without a clear recorded score.

3.2 Using PageRank

PageRank is uses an iterative process to distribute a constant total value amongst the nodes of a graph. At each step, the current PageRank of a node is redistributed equally via each of its outedges. To resolve edge cases, at the end of each step the value of each node is dampened by a factor of s, and the resulting $1-s$ is redistributed uniformly to the nodes.

We used the `pagerank` method supplied by `networkx`, with the default dampening value of 0.85.

4 Comparison with available rankings

The Elo rankings were obtained from https://statsmash.io/elo/.

Player	PageRank	Elo
Dabuz	1	8
Light	2	11
MkLeo	3	1
Marss	4	2
Tweek	5	3
Nairo	6	5
Samsora	7	6
Glutonny	8	12
ESAM	9	13
MVD	10	31
Cosmos	11	18
Shuton	12	7

There are some notable discrepancies in the rankings: the two that stand out the most are Dabuz and MVD. Further analysis will be needed to develop explanations.

5 Looking forward

The rest of the project will be devoted towards developing metrics to compare the two rankings, and to understand why certain discrepancies exist. Issues we would like to explore include:

- Artefacts from the cleaning process: part of our cleaning process involved removing games with incomplete data. This resulted in the creation of components disconnected from the main graph. Many of these components are of size two, indicating that they happened in the second round; we can consider how re-adding these games would change the rankings.
- Clusters: the data was taken from hundreds of tournaments across different countries and continents. We can analyze the significance of having such clustered and disjoint populations. How do regional rankings (e.g. the Japanese circuit and the North American circuit) compare to the global rankings?
- Weighted edges: in this preliminary model, each edge represented a game, weighted equally. However, our data also includes the severity of the win; if we weighted the edges accordingly, would we arrive at a different result?

6 Appendix

6.1 clean.py

```
# clean.py - Parses the data set and outputs 3 files
# - clean.csv: player ids and set scores
# - player_ids.csv: player names and ids
# - tournament_attendance.csv: tournament names and # players

import csv

desired_fields = ['winner_global_id', 'loser_global_id',
                  'winner_score', 'loser_score']
extra_fields = ['winner_name', 'loser_name', 'tournament_name']
field_indices = {}
players = {}
tournaments = {}

# check if entry has valid data
def isValid(line):
    # check all validation rules
    if line[field_indices['winner_global_id']] == '0': return False
    if line[field_indices['loser_global_id']] == '0': return False
    if line[field_indices['loser_score']] == '-1': return False
    if line[field_indices['loser_score']] == '': return False
    if line[field_indices['winner_score']] == '0': return False
    if line[field_indices['winner_score']] == '': return False
    return True

# save players and their ids
def savePlayers(line):
    # retrieve values
    winner_id = int(line[field_indices['winner_global_id']])
    loser_id = int(line[field_indices['loser_global_id']])
    winner_name = line[field_indices['winner_name']]
    loser_name = line[field_indices['loser_name']]

    # save winner
    if winner_id not in players:
        players[winner_id] = [winner_name]
    elif winner_name not in players[winner_id]:
        players[winner_id].append(winner_name)
```

```
    # save loser
    if loser_id not in players:
        players[loser_id] = [loser_name]
    elif loser_name not in players[loser_id]:
        players[loser_id].append(loser_name)

# keep track of the tournaments and number of attendees
def saveTournament(line):
    tournament_name = line[field_indices['tournament_name']]
    if tournament_name not in tournaments:
        tournaments[tournament_name] = 0
    tournaments[tournament_name] += 1

# combine information for desired fields
def getInfo(line):
    obj = {}
    for x in desired_fields:
        obj[x] = line[field_indices[x]]
    return obj

# read the data from file
with open('ultimate_sets.csv') as sets_data:
    with open('ultimate_sets_clean.csv', mode='w') as cleaned:
        # find where desired fields are in file
        data_reader = csv.reader(sets_data)
        headers = data_reader.next()
        for i, header in enumerate(headers):
            if header in desired_fields:
                field_indices[header] = i
            if header in extra_fields:
                field_indices[header] = i

        # write header to file
        writer = csv.DictWriter(cleaned, fieldnames=desired_fields)
        writer.writeheader()

        # process each line
        for line in data_reader:
            if not isValid(line): continue
            savePlayers(line)
            saveTournament(line)
            writer.writerow(getInfo(line))
```

```
# write player and id information
with open('ultimate_player_ids.csv', mode='w') as player_ids:
    writer = csv.DictWriter(player_ids, fieldnames=['id', 'player'])
    writer.writeheader()
    for player_id in sorted(players.keys()):
        writer.writerow({
            'id': player_id,
            'player': ' [or] '.join(players[player_id])
        })

# write tournament name and attendance
with open('ultimate_tournament_attendance.csv', mode='w') as out:
    writer = csv.DictWriter(out, fieldnames=['tournament', 'count'])
    writer.writeheader()
    for tournament in sorted(tournaments.keys()):
        writer.writerow({
            'tournament': tournament,
            'count': tournaments[tournament]
        })
```

6.2 pageranker.py

```
# pageranker.py - Represents the PageRanking of a
# set of players given their game set

import pandas as pd
import networkx as nx
import numpy as np
import operator
import csv

class PageRanker:
    games = ""
    names = ""

    def __init__(self, games, names):
        self.games = pd.read_csv(games) # 6 column cleaned data
        self.names = pd.read_csv(names) # 2 column cleaned data

    def build_edgelist(self):
        # data with only IDs
        data2 = self.games[['winner_global_id', 'loser_global_id']]
        edgelist = []
```

```
        for i in range(0, len(data2)):
            edgelist.append(str(data2['winner_global_id'][i]) + ' '
                        + str(data2['loser_global_id'][i]))
        return data2

    def build_digraph(self, edgelist):
        g = nx.DiGraph()
        for i, row in edgelist.iterrows():
            g.add_edge(str(row[1]), str(row[0]),
                attr_dict=row[2:].to_dict())
        return g

    def pagerank(self, g):
        return nx.pagerank(g)

    def pagerank_sort(self, pr):
        return dict(sorted(pr.items(),
            key=operator.itemgetter(1),reverse=True))

    def pagerank_write(self, pr):
        f = csv.writer(open('pageranks.csv', 'w'))
        f.writerow(['ID', 'PageRank'])
        for key, val in pr.items():
            f.writerow([key, val])

    def id2name(self, id):
        try:
            return self.names['player'][list(names['id'][0:]).index(id)]
        except ValueError:
            return -1

    def name2id(self, name):
        try:
            return self.names['id'][list(names['player'][0:]).index(name)]
        except ValueError:
            return -1

    def leaderboard(self, n):
        ranks = pd.read_csv('pageranks.csv')
        print("Leaderboard:")
        board = np.array([['Rank', 'Player', 'ID']])
        for i in range(0, n):
            board = np.append(board, [[i, self.id2name(ranks['ID'][i]),
```

```
                str(ranks['ID'][i])]], axis=0)
        return board

    def id2rank(self, id):
        ranks = pd.read_csv('pageranks.csv')
        return list(ranks['ID']).index(id)
```