

# Tier List Ranking: Novel Round Based Ranking

Chris Sisemore



**Abstract**—This paper provides an in depth explanation of the novel *tierRank* algorithm, including its structure and functionality. Also contained are trivial and realistic test cases with analyses, with additional hypotheses and future development areas. cursory background on relevant ranking algorithms, and the foundational ideas and terminology of tier list generation are added to drive further research into this algorithm and relate its core ideas to the rest of the ranking field.

## 1 INTRODUCTION

DESPITE being a relatively unknown concept in computer science, tier lists are useful for general quantification and fit neatly into existing and popular ranking paradigms. They can be explained simply as ordered lists, which divide their contents into subsections labelled 'tiers'. Players of competitive fighting games invented tier lists to help understand which game characters are optimal choices in professional or skill competitive one-on-one battles[8]. Similar subjective rankings exist for companies and universities, which may be more familiar to readers. This paper provides an algorithm which models the thought process players use to create tier lists, in addition to a complete background on both tier lists and ranking, which may be skipped if the reader is only interested in the algorithmic explanation.

## 2 RANKING BACKGROUND

### 2.1 The fundamental idea of ranking

Ranking is essentially the problem of finding a score for a set of objects, where a higher score indicates more usefulness of some kind. Some of the algorithms discussed in this background have functionality in addition to this, particularly the discovery of objects in networks[2].

### 2.2 Depth of Research

Ranking problems and their algorithmic solutions have been a fundamental piece of computer science development for decades and their most prominent uses have been in data retrieval, most notably search [1]. For example, deciding which entries to show from a database which has hundreds of entries that, on the surface, match the query requirements but in depth are not of equal value to most searchers. Solutions to this particular problem are the most researched and discussed.

## 3 GENERAL RANKING APPLICATIONS

### 3.1 Search

As previously mentioned, search is the most commonly discussed ranking application[1][2][3].

#### 3.1.1 Internet

Google's search results, which is to say the mapping and ranking of URLs on the World Wide Web, are the most prominent version of this application. This also includes searching inside of larger websites. The main difference between the larger internet search application and the following recommendation applications is that they tend to have some function that also helps to find and index pages. While Google search is the most famous use case, a close second would be Facebook friend recommendations, and following that would be any of the various front pages of Twitter, Youtube, or content aggregation site which are born of similar query nature. These queries are based on

user history or provided preferences to decide the ranking and they draw from large databases more frequently than indexed networks[2].

### 3.2 Prediction

These applications usually have a smaller and more easily definable set of objects which to rank. Often those objects are found through simple data matching queries, which are unranked, but sometimes they're a result of the previous application.

#### 3.2.1 Sports Ranking

Ranking of players in sports are important to the prediction of future performance which is in turn useful for coaches determining how to select players for their teams, and for gambling firms who must decide who to bet on individual games. Additionally, if there are multiple tournament settings with no overall winner for that year in any given sport, ranking will be used to declare the best player. The most famous use of this style of ranking is in tennis [6].

#### 3.2.2 Language Recognition

Consider the situation where you have several possible implications or meanings for a given word when trying to understand overall sentence meaning. Ranking is used to decide which meaning is the most likely to be correct[3].

## 4 STUDIED METHODS

### 4.1 Graph Algorithms

Graph based algorithms are the foundation of most modern rankings. The most prominent of them is Google's original PageRank. PageRank works by picking an arbitrary starting web page (node), finding all the links on that web page, and then surfing to those pages. Each page starts with a rank of zero, and each time it is linked to that it will receive an extra point of rank. This algorithm will essentially find all connected pages and give the highest rank to those which are the most commonly linked to. Of course, this algorithm, once known, is easily taken advantage of by sites which want

to increase their rank; so, in the current version, there are surely other ranking systems being used to identify such sites[2]. Despite that, when people cannot modify their nodes, this ranking is still rather effective. It has been used as an alternative to traditional sports rankings, where the player is treated as a node with links to all the players they have played against, adding rank for wins and removing it for losses.[6]. There are many alternative versions of this algorithm, which modify the rank based on different nuances, but they are similar in kind.

### 4.2 Listwise Ranking

Listwise ranking is basically the creation of a function that takes in certain features of an object and, based solely on those features generates a rank value. It has been shown that when there isn't excellent pairwise data and this function is well tuned, it can be the most effective way of ranking relatively small groups of objects [4].

### 4.3 Pairwise Ranking

Pairwise ranking is finding rank scores based on comparisons between objects of a limited set. Rank scores are, therefore, usually a summation of scores from a set of comparisons between nodes which, in application, can be very similar to graph algorithms, assuming each edge of a graph is a pairwise comparison. However, in most uses of pairwise ranking there is a comparison matrix, which has values for each comparison of each object to each other object. These matrices can be complete, including all the comparisons, or incomplete, missing some comparisons, while still producing satisfactory rankings. When using a comparison matrix, this problem can be broken down into a complex linear algebra equation. Depending on how you weight and use this matrix, you can modify the rank values[5]. Alternatively, the comparisons can be done using complex functions or round based games and tournaments, but these haven't been deeply explored, though their development has increased in recent years [4].

### 4.3.1 Machine Learning

Machine learning techniques are now applied in all of the previously mentioned algorithms, particularly in pairwise comparison functions and listwise ranking functions. It turns out developing functions for these, which create useful rankings, is beyond human comprehension, so neural networks and boost decision trees are trained on real world data, AB testing etc. Since these machine learning techniques are so recent, the non-matrix based pairwise and all listwise ranking algorithms are relatively unexplored and underdeveloped[4].

## 5 TIER LIST BACKGROUND

The history of tier lists isn't especially long or noteworthy; however, because they are nested in a particular sub-culture, there exists a bit of relevant jargon and standard formatting. This is not at all required to understand the ranking algorithm derived from them, but is useful for interacting with them in a broader sense.

### 5.1 Tier Names

Most important are tier names. In the typical tier list there are six tiers, depending on the number of characters being ranked. Some creators organize inside of tiers, such that the first character in that tier is the best and the last is the worst.

#### 5.1.1 Letter Naming

The most common labeling is "S, A, B, C, D, F". Many English speakers will recognize the final five as the letter grading system for students, but will be confused by the 'S'. This 'S' comes from the Japanese grading system where 'S', in place of 'A', is the highest letter grade. The Japanese grading system became the standard because fighting games are largely popular in Japan, most fighting games are developed by Japanese companies, and, in said games, there are sometimes ranking systems which use these letters. To create additional tiers, players also modify the letters with '+' or '-', again just like letter grades, e.g. 'B-' or 'A+'. Additionally, letters may be added to create more tiers,

explained by example: 'SSS' is better than single 'S' which is better than 'AAA' which is better than 'A'. If you had 'SSS', 'SS', and 'S', they could be mapped to 'S+', 'S', and 'S-' respectively. The table at 5.1.1 depicts a very well delineated lettered tier list.

TABLE 1  
Detailed Letter Naming Paradigm

S+	SSS
S	SS
S-	S
A+	AAA
A	AA
A-	A
B+	BBB
B	BB
B-	B
C+	CCC
C	CC
C-	C
D+	DDD
D	DD
D-	D
F	F

#### 5.1.2 Relative Naming

Relative naming is the second most common naming paradigm and is straightforward. 'Top' tier is the best, then 'high' tier, 'mid' tier, 'low' tier, and 'bottom' tier. The tiers, besides top and bottom, can be further modified with low and high, e.g. 'low-mid' tier.

TABLE 2  
Map from Relative Naming to Letter Naming

Top	S
High	A
High-Mid	B
Mid	C
Low	D
Bottom	F

### 5.2 Comparison Matrices

Tier list creators usually don't think of these as matrices and, instead, call them 'matchup

charts'. Typically they are organized around a single character and ranking how hard it is for them to beat every other character. If you have a matchup chart for every character you can create something like figure 7.

## 6 THE NOVEL ALGORITHM

Now that the reader has a comfortable background in what tier lists and ranking are, consider how a player might make a tier list. Most people base their list on matchup charts, such that whichever character has the best overall matchups wins; but, to figure out who has the best matchups it is not enough to simply find the number of matchups that a character wins. To understand why, consider the example of Rock Paper Scissors Well.

### 6.1 Rock Paper Scissors Well Example

First we assume that, as is obvious in normal Rock Paper Scissors, all options are of equal value. Then Well is added. Well beats Scissors and Rock, but loses to Paper. Now we can clearly divide them into the groups Well and Paper, which each beat two other options, and Rock and Scissors which each beat one other option. Between Paper and Well, Well beats the two lower tier choices, and Paper beats one lower tier choice and one upper choice, so Paper has better matchups, and is therefore better than Well. The same logic applies to Scissors and Rock, who both only beat one other option, but Scissors beats a higher tier option and is therefore better than Rock. Following this logic, we have a complete ordering of Rock Paper Scissors Well.

### 6.2 Formalizing The Example

By considering each character (Rock, Paper, Scissors, and Well) as an object with a ranking value, *Self - Worth*, and then defining a round as the time every single character updates their *Self - Worth* based on the *Self - Worth* of their matchups, we can draw a diagram for the example. Note, we use one as the initial *Self - Worth* arbitrarily, as the number doesn't matter.

S				
A				
B				
C				

Fig. 1. Rock Paper Scissors Well Example: Round 1





S		
A		
B		
C		

Fig. 2. Rock Paper Scissors Well Example: Round 2

### 6.3 In Algorithmic Terms

Define *char* as an array of all characters in a given game as objects with *self\_worth*, and *new\_self\_worth* variables. Also require them to possess a function, *battle*, that returns their win percentage against any other character. Then, for the first round, set all characters' *self\_worth* to some arbitrary constant. For each round, set the *new\_self\_worth* to 0 for all *char*. Then for each character in *char* use the sum *battle\*all other characters.self\_worth* to define each character's *new\_self\_worth*. Use the sum of all character's *new\_self\_worth* divided by the number of characters to normalize the individual characters *new\_self\_worth*. Proceed to set *self\_worth* equal to *new\_self\_worth*, and move to the next round for the desired number of rounds. It may be viewed as executable code in the same Git folder as this report.

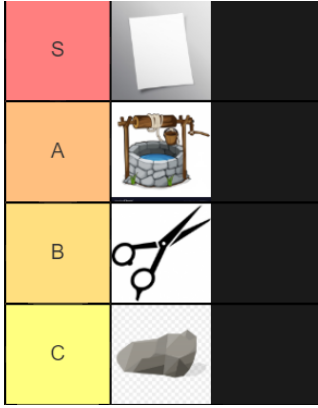


Fig. 3. Rock Paper Scissors Well Example: Round 3

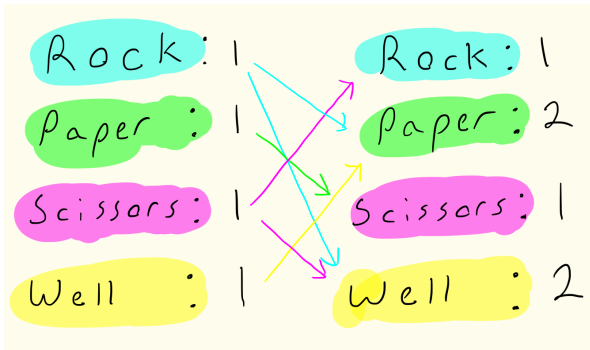


Fig. 4. Round 1 to Round 2

## 7 HOW TO RUN THE PROVIDED ALGORITHM

To run the included implementation of this algorithm, first have a Python3 compiler with the *numpy* and *matplotlib* packages installed. Given this, start a Python3 environment in this folder. Import *tierRanker* and run *tierRanker.buildList()*. From there you will be prompted to provide the prerequisite data for the algorithm with text prompts. Simply follow those prompts. Once the program has all the necessary data it will run the algorithm with four round sizes (10, 20, 50, 100) and output graph images for each like figure 5.

## 8 TESTING AND RESULTS

### 8.1 Rock Paper Scissors Well Real Runs

As basic test of this algorithm and its implementation, it was run with the Rock Paper Scissors Well (RPSW) example. First where the values for *battle* were only ones or zeros,

zero being always losing and one being always winning. This run can be seen in figure 5. Then, to see how much the algorithm's behavior changed when using partial wins, as represented by decimals from zero to one, winning was treated as 0.75 and losing was treated as 0.25, e.g. Rock against Scissors is 0.75 and Scissors against Rock is 0.25. This run is shown by figure 6.

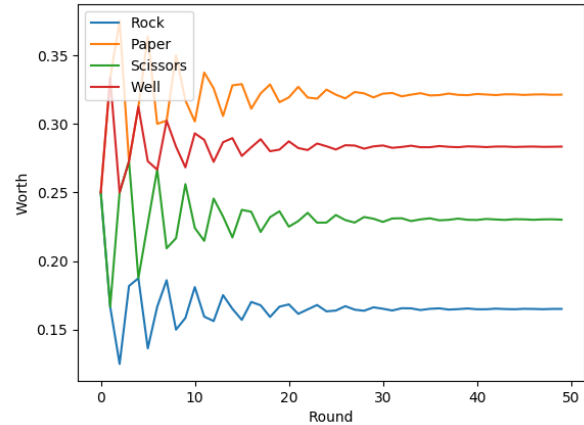


Fig. 5. Rock Paper Scissors Well Self-Worth Values Normalized Over Rounds

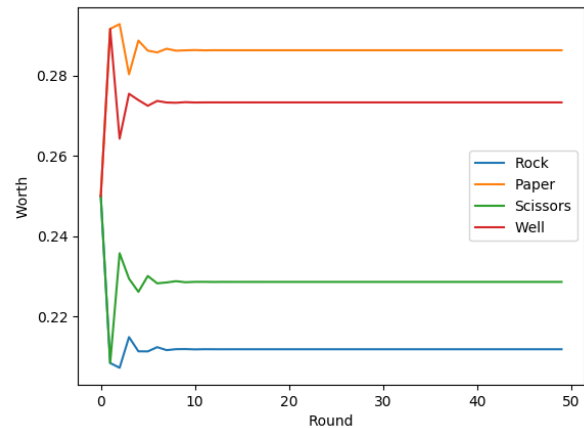


Fig. 6. Rock Paper Scissors Well Self-Worth Values Over Rounds with Partial Victory

### 8.2 Smash Brothers Melee

This game was chosen as the example because it has a relatively small number of characters, well explored matchups, and a long history of

tier lists. Figures 7 and 8 are the data used to run this test. Table 8.2 shows how it was translated into decimal values for the algorithm and the results are in figure 9.

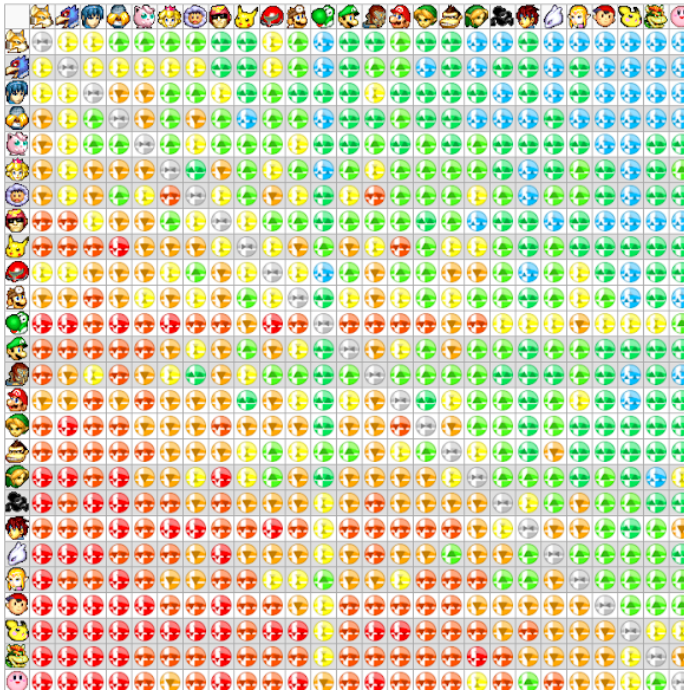


Fig. 7. Smash Bros Melee Matchups [9]

Icon	Description
	Large disadvantage
	Disadvantage
	Slight disadvantage
	Even
	Slight advantage
	Advantage
	Large advantage
	Mirror Match

Fig. 8. Key for Matchup Chart

## 9 ANALYSIS AND NEXT STEPS

Overall, the tests of this algorithm proved to find some consistent ranking when given matchup and character data, which means that the algorithm has successfully modeled at least

TABLE 3  
Map from Melee Matchups to Algorithm *battle* values

Large Disadvantage	0.2
Disadvantage	0.3
Slight Disadvantage	0.4
Even	0.5
Slight Advantage	0.6
Advantage	0.7
Large Advantage	0.8
Mirror Match	0.5

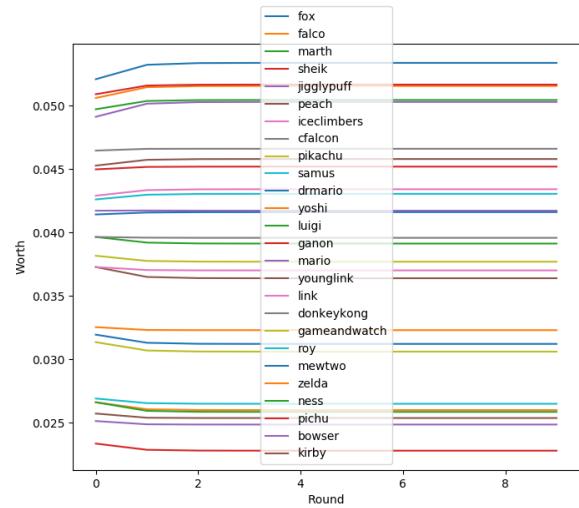


Fig. 9. Results for Smash Brothers Melee Testing

some form of a tier list thought process. However, there are notable differences between each test and many possibilities to further develop and research this style of ranking algorithm. This section explores what those might be.

### 9.1 Notable Differences Between Each Test

The first test, figure 5, which has few characters and well defined win and loss conditions, shows by far the most osculation of all of our tests. The other two tests both arrive at their final orderings more quickly. We speculate that the more extreme the win and loss values are, the more osculation will occur before a stable ranking forms. Especially in cases with few characters. Further tests on this existing algorithm with varied games would be worthwhile, and would explore this hypothesis. Additionally, it appears all the tests need very few



rounds to find useful rankings. If this is the case even with sets of much larger characters, it may be worth exploring the exact average number of rounds necessary to find the final ranking. If it were few enough, then this algorithm could be considered relatively efficient.

## 9.2 Modifying the Algorithm

None of this data has been compared with other existing subjective tier lists. It would be interesting to survey people and use their subjective matchup charts for any number of games to compare to their overall tier lists and see if the algorithm creates the same list, possibly testing whether more extreme or less extreme *battle* function values more accurately reflect human thought.

### 9.2.1 How To Create the *battle* function

In these tests, the battle function is based off of well agreed on subjective matchup charts, or objectively known matchup charts in the case of Rock Paper Scissors Well; however, this function could be created in a myriad of ways. First as it is now, from any comparison matrix for all the objects, but additionally, one could use an incomplete comparison matrix which contains rules for handling unknown battles. Alternatively, some machine learning algorithm could be used for this function, trained on many similar comparisons with known winners and losers, or even straight statistics could be used. These would produce different and interesting lists which may be useful in other applications[5].

### 9.2.2 Defining Tiers

It may also be relevant to develop some equation which could group tiers and create tier divisions. In figure 9, we see several clear groupings that could be cut into tiers, but creating a statistical model for the borders of tiers is an interesting development and research area.

## 9.3 Comparing Other Ranking Methods

Using similar inputs you could compare the rankings created by this algorithm and any

other ranking algorithm mentioned in the ranking background. Comparing them for both tier list creation and general ranking applications should be possible.

## 10 CONCLUSION

This algorithm, *tierRanker*, has proven to model a subjective human process of making tier lists for fighting games, and a general process for ranking objects. There is still much research and development possible to find interesting scientific conclusions about how tier lists work and how this algorithm can be used to model them. The field of ranking in computer science has a breadth of algorithmic solutions which can be used with this algorithm to synthesise further hypotheses and experimentation. Using the knowledge in this document, hopefully further scientific exploration of this particular phenomenon will occur.

## ACKNOWLEDGEMENTS

I would like to thank those my TA for grading this paper. I would also like to thank Nick Dooley and Austin Martin for proofreading.

## REFERENCES

- [1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," 11-Nov-1999. [Online]. Available: <http://ilpubs.stanford.edu:8090/422/>. [Accessed: 26-Feb-2020].
- [2] P. de Keyser, "11 - Indexing the web," in Indexing, P. de Keyser, Ed. Chandos Publishing, 2012, pp. 195–219.
- [3] R. Mihalcea, "Graph-based ranking algorithms for sentence extraction, applied to text summarization," in Proceedings of the ACL 2004 on Interactive poster and demonstration sessions -, Barcelona, Spain, 2004, pp. 20-es, doi: 10.3115/1219044.1219064.
- [4] "Learning to Rank: From Pairwise Approach to Listwise Approach," in Proceedings of the Twenty-Fourth International Conference, Corvallis, Oregon, USA, 2007, doi: 10.1145/1273496.1273513.
- [5] N. M. Tran, "Pairwise ranking: Choice of method can produce arbitrarily different rank order," Linear Algebra and its Applications, vol. 438, no. 3, pp. 1012–1024, Feb. 2013, doi: 10.1016/j.laa.2012.08.028.
- [6] S. Bozoki, L. Csató, and J. Temesi, "An application of incomplete pairwise comparison matrices for ranking top tennis players," European Journal of Operational Research, vol. 248, no. 1, pp. 211–218, Jan. 2016, doi: 10.1016/j.ejor.2015.06.069.
- [7] "Tier Lists for Everything," TierMaker. [Online]. Available: <https://tiermaker.com/>. [Accessed: 10-Mar-2020].

[8] "Tier list," Wikipedia. 19-Feb-2020.

[9] "Character matchup (SSBM)," SmashWiki. [Online]. Available: [https://www.ssbwiki.com/Character\\_matchup\\_\(SSBM\)](https://www.ssbwiki.com/Character_matchup_(SSBM)). [Accessed: 14-Mar-2020].