PREDICTING AND ENHANCING HEARTHSTONE STRATEGY WITH COMBINATORIAL FUSION

BY

Henry William Gorelick

BA, University of Michigan, 2017

MASTER'S THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE MASTER OF SCIENCE DEGREE

IN THE DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

AT FORDHAM UNIVERSITY

NEW YORK

MAY, 2020

Acknowledgement

First and foremost, I would like to thank Dr. D. Frank Hsu for his guidance, expertise, and patience throughout the process of writing this master's thesis. While his combinatorial fusion analysis (CFA) methodology had been applied to numerous fields including medicine, business, technology, sports, and various fields of scientific research, he had not yet used it for games and game playing. Not only was gaming a new field for Dr. Hsu and CFA, but Hearthstone is not an easy game to learn for somebody who lacks experience. I am incredibly thankful to Dr. Hsu for enduring the many long meetings, miscommunications, and disagreements it took for us to complete this work. He took a risk in agreeing to work with me on this project, and for that I am deeply grateful.

Additionally, I would like to thank Dr. Truong-Huy D. Nguyen for introducing me to the Hearthstone research community. This project began with him and our brief time working together on applying Monte Carlo Tree Search to Hearthstone. Also, I would like to thank Dr. Amy K. Hoover for encouraging me to continue this project when I thought I would not be able to return to it, and for her critical role in convincing Dr. Hsu to sign on as my advisor.

I would also like to thank the committee members for their role in this thesis: Dr. Mohammad Ruhul Amin, Dr. Ying Mao, Dr. Tadeusz Strzemecki, and Dr. Yijun Zhao. Whether it was in the classroom or in research meetings/assignments, they taught me the foundational concepts of data science, AI, machine learning, and research required to complete this project.

I also give my sincerest thanks to Ajmal Aqtash for his constant support, enthusiasm, and encouragement. Not only is Ajmal one of the greatest mentors and teachers I have had the chance to work with, but he is also a great friend.

Finally, I would like to thank my family, friends, and loved ones for giving me the time and space to complete this work, and for giving me support and encouragement throughout.

Table of Contents

1 Introduction

Games embody the human experience. They test our ability to learn and improvise, our imaginations, and our intellect. They entertain us and push our instinctual sense of competition. We can consider the search for general Artificial Intelligence (AI) a game itself as a solution will epitomize these same themes that are central to what it means to be human. But measuring the human experience is difficult. Instead, if we can explore and design computational systems that can beat humans at games, then we can discover answers to questions about human intelligence. Furthermore, through solving game-related problems we can learn something new about issues in real-life like planning [1] and decision making [2]. And so, the histories of games and AI are expectedly intertwined.

1.1 AI, Games, and Hearthstone

1.1.1 Brief History of AI and Games

The modern history of AI and games can be traced back to 1928 when John von Neumann proved the minimax theorem. Minimax is an intuitive game-playing strategy and states that, "…you should always choose a move such that, even if the opponent chooses the absolute best response to that move, and to each of your future moves, you will still get the highest score possible at the end of the game," [3]. Twenty-two years later, in 1950, *Philosophical Magazine* published Claude Shannon's groundbreaking paper on a computer program that uses the minimax algorithm to play chess [4]. Shannon's work is widely regarded as the first to apply computer AI methods to games [3]. Then in 1997, twenty-seven years after Shannon's original work, IBM's Deep Blue computer famously defeated chess world champion Garry Kasparov and announced a new era of superhuman AI [5]. Finally, and most recently, in 2016, DeepMind's AlphaGo—which uses deep computational neural networks and Monte Carlo Tree Search—

conquered the combinatorically explosive game of Go: a single game of Go has 250^{150} possible move sequences. AlphaGo defeated the European Go champion Fan Hui five times in five games [6].

1.1.2 Hearthstone

AlphaGo's success is undoubtedly one of the greatest achievements in modern AI and Machine Learning (ML) research. However, AI and ML still struggle to master turn-based strategy games that add characteristics such as nondeterminism and partial information to an already combinatorically complex state-action space. Hearthstone, a collectible card game by Blizzard Entertainment for PC and mobile, packages these characteristics in manner that is easy to learn, but hard to master, and therefore has recently become a testbed for applying AI and ML techniques to games [7].

Enter, Hearthstone. Hearthstone is a turn-based collectible card game between two players. Players choose from nine heroes, each having unique cards and abilities, and construct a deck of thirty cards. During a match, players spend mana crystals to play cards and damage their opponent with the goal of reducing the opponent's health to zero. See *Appendix A* for a brief guide to Hearthstone's rules and mechanics.

1.1.3 Formal Game Description

Since Blizzard has not published an official rulebook for Hearthstone, players have taken it upon themselves to create a full, unofficial rulebook explaining the game's mechanics and processes [8]. Furthermore, Andersson and Hesselberg provide a formal description of Hearthstone as a problem for AI research [9]:

- **Imperfect Information** – Hearthstone relies on uncertainty and limiting players' access to certain information throughout the game. Players cannot see their opponent's hand or deck until cards are played or shown through certain game mechanics (i.e. drawing a card with a full hand). Furthermore, players' decks are shuffled prior to play, thus hiding the order of cards in players' decks. At the end of a game, any cards that have not been played (cards remaining in players' hand and deck) are not revealed to the opponent. This forces players to anticipate various possibilities and predict opponents' moves.

- **Stochastic Outcomes** – Hearthstone heavily utilizes randomness. In certain situations throughout a duel, the stochastic nature of the game can make outcome prediction exceedingly difficult.
- **Complexity** – When we performed the experiments presented in this thesis, Hearthstone had 2,996 playable cards, and Blizzard releases roughly 140 new cards every three or four months [10]. The number of cards, the number of combinations of cards that could be included in a single deck are just one aspect of Hearthstone's complexity. Separately, a single turn of Hearthstone is incredibly complex as we will explain in *Section 2.1*.
- **Zero-Sum** – Hearthstone is a two-player game, and only one can win.
- **Turn-Based** – A Hearthstone duel is inherently discretized through players taking alternating turns until the game's end.
- **Finite** – A single turn in a Hearthstone match has a 75 second time limit. Though players can elect to end their turn at any time before the limit is reached. And a game of Hearthstone ends in a draw if it reaches the 90^{th} turn.

1.2 Machine Learning, AI, and Combinatorial Fusion Analysis (CFA)

Modern scientific inquiry and knowledge discovery processes have faced a complex and challenging environment. It is an interconnected, data-centric, information-based, cyber-physical-natural ecosystem. For example, a variety of sensor and imaging technologies have generated large and diverse data in domains ranging from biomedicine, social networks, to economics, climate change, health care, and cybersecurity. Researchers and professionals have found the big data deluge and information overload to be challenging [11].

Machine learning and AI has seen fundamental advances in support vector machines with the kernel method and the AdaBoost ensemble methods [12, 13]. Ensemble and data fusion methods have been used to combine decision trees, pattern classifiers, information retrieval systems, molecular similarity measures, and artificial neural nets [14, 15, 16, 17, 18, 19]. More recently, combinatorial fusion, a paradigm similar to ensemble method and data fusion, was proposed and developed to combine multiple scoring systems (MSS) using rank-score characteristics (RSC) functions and cognitive diversity [20, 21, 22, 23]. Combinatorial fusion, and the general combination of multiple scoring systems has been used in various domains including science, technology, society, and business such as target tracking, virtual screening, cognitive neuroscience, ChIP-seq peak detection, portfolio management, and similarity ranking [24, 25, 26, 27, 28, 29].

In combining multiple scoring systems, combinatorial fusion considers both score and rank combinations. Score combination operates on the parametric Euclidean score space, while rank combination operates on the non-parametric permutation rank space. The relation between rank and score values of a data item by a scoring system is defined as the rank-score characteristic (RSC) function [20, 21, 30, 23]. Hsu and Taksa [22] provides a condition, involving cognitive diversity, under which rank combination performs better than score combination.

1.3 Improving Hearthstone Game Strategy with Model Fusion using CFA

Model fusion, as a special case of combinatorial fusion where each model is considered as a scoring system, combines diverse and “good enough” models to achieve better results in forecasting, prediction, or analytics. In this thesis, we use multiple machine learning models to analyze a dataset of 500 Hearthstone games with a set of features determined by previous game experience and expert systems. Each model produces a scoring system with a score function and a rank function. The RSC function is then used to characterize the ranking (or scoring) behavior of the model. Each of the combinatorial combinations of these models is then evaluated using precision criterion.

Let $D = \{d_1, d_2, \dots, d_n\}$ be a set of n games. Let A be a model, or scoring system, of the dataset D which assigns a score to each of the games d_i in D. The score function $s_A : D \rightarrow \mathbb{R}$, assigns a real number in $\mathbb{R}$ to each game d_i in D. The rank function $r_A : D \rightarrow N$, where $N = \{1, 2, \dots, n\}$ and $n = |D|$, is obtained by sorting the score values into descending order and assigning a rank order of the score value to the game having that score value. For a scoring system A with scoring function s_A and its corresponding rank function r_A, the rank-score characteristic (RSC) function $f_A : N \rightarrow \mathbb{R}$ was defined by Hus, Shapiro, and Taksa as the following formula [20, 21, 30, 23]:

$$f_A(i) = s_A({r_A}^{-1}(i)) \text{ for } i \text{ in } N$$

Equation 1.1: Rank-Score Characteristic (RSC) Function f_A

For two models A and B, score functions s_A and s_B, and rank functions r_A and r_B, the score function of the score combination (SC) and the score function of the rank combination (RC) of the two scoring systems are defined as:

$$s_{SC}(d_i) = \frac{s_A(d_i) + s_B(d_i)}{2}$$

Equation 1.2: Score Function for Score Combination SC(A, B)

$$s_{RC}(d_i) = \frac{r_A(d_i) + r_B(d_i)}{2}$$

Equation 1.3: Score Function for Rank Combination RC(A, B)

Furthermore, the rank functions of the SC and RC, r_{SC} and r_{RC}, can be derived accordingly. For models A and B and RSC functions f_A and f_B, respectively, the cognitive diversity, $CD(A, B)$, is defined as:

$$CD(A, B) = \sqrt{\sum_{i=1}^{n} (f_A(i) - f_B(i))^2}$$

Equation 1.4: Cognitive Diversity of Two RSC Functions in terms of RSC Functions f_A and f_B

1.4 Overview

In this master's thesis, we demonstrate that model fusion using combinatorial fusion analysis (CFA) can accurately predict the winner of a game of Hearthstone. Chapter 2 begins by providing the empirical analysis of Hearthstone's complexity, which segues into an outline and comparison of previous works that have applied AI and ML to Hearthstone. This section gives context to our work within the current research landscape and presents the logic behind our search for a different method of using AI and ML for Hearthstone. The chapter concludes with the basics of using CFA for model fusion and the techniques we used for evaluating fusion performance.

Chapter 3 highlights the simulation environment we used as the foundation for constructing and implementing our system. We continue with details of our methods for dataset construction and feature selection. Finally, Chapter 3 wraps up the specifics of performing CFA on Hearthstone data and model selection. Chapter 4 provides our experiments and results. This thesis concludes with Chapter 5 where we summarize our conclusions and provide examples of how we could continue to build upon these conclusions in the future.

2 Hearthstone's Complexity and Model Fusion

We can illustrate the mathematical support for the use of AI and ML techniques to predict and improve Hearthstone strategies. Then, with the game's complexity in mind we present some of the existing methods for applying AI and ML to Hearthstone. Subsequently, we describe the specifics of combinatorial fusion analysis (CFA) for model fusion and the techniques employed for evaluating the system's performance.

2.1 Hearthstone's Theoretical State-Action Space Complexity

The size of the state-action space for a game of Hearthstone is one of the key difficulties in predicting the outcome of a game of Hearthstone and/or developing an effective move selection algorithm. In fact, the complexity of a single Hearthstone turn makes developing an effective, rule-based AI agent completely unreasonable. We can demonstrate this issue by finding the maximum $\boldsymbol{N}$ possible unique sequences of moves a player could execute before ending their turn. We find $\boldsymbol{N}$ by calculating the product of the following:

- The $\boldsymbol{A}$ possible attacking moves
- The $\boldsymbol{T}!$ different orders in which a player can execute $\boldsymbol{T}$ available attacks
- The $\boldsymbol{M}^{\boldsymbol{P}}$ ways $\boldsymbol{P}$ cards in a player's hand can be played on $\boldsymbol{M}$ targets
- The $\boldsymbol{P}!$ different orders in which $\boldsymbol{P}$ cards can be played

Generally, $\boldsymbol{A}$ is calculated as the number of opponent's attackable characters raised to the count of a player's characters that can attack. We find the value of $\boldsymbol{A}$ according to Hearthstone's rules and mechanics such as which minions are exhausted, frozen, have taunt, etc. $\boldsymbol{T}$ is the total number of attacks available to a player. Finally, like $\boldsymbol{A}$, the rules of the game are considered when determining $\boldsymbol{M}$ and $\boldsymbol{P}$. Together, the formula for $\boldsymbol{N}$ is defined as:

$$N = A \times T! \times M^P \times P!$$

Equation 2.1: Theoretical Maximum Hearthstone Turn Complexity

With maximum values of $\boldsymbol{A} = 8^8$, $\boldsymbol{T} = 30$, $\boldsymbol{M} = 16$, and $\boldsymbol{P} = 10$ (derivations explained in *Appendix B*), the maximum value of $\boldsymbol{N} \approx 1.78 \times 10^{58}$. It is important to note, however, that *Equation 2.1* does not account for cards with random outcomes. Factoring in these types of cards would exponentially increase $\boldsymbol{N}$. Also, technically speaking, there are possible scenarios in which $\boldsymbol{P}$ approaches $+\infty$.

While the probability of $\boldsymbol{N}$ reaching this maximum value is infinitesimal, we must consider it as the theoretical upper bound. In most real game situations $\boldsymbol{N}$ is drastically smaller. As moves are executed during a player's turn, the values of $\boldsymbol{A}$, $\boldsymbol{T}$, $\boldsymbol{M}$, and $\boldsymbol{P}$ will almost always decrease as a result of spending mana crystals, the cost of the cards the player's hand, the death of the player's and opponent's minions, etc.

2.2 Existing Methods

Since the recent successes of DeepMind's AlphaGo [31] and AlphaZero [32], the use of games for artificial intelligence and machine learning research has dramatically increased. And, a substantial number of researchers have published the results of applying different methods for game playing AI—including DeepMind's own method—to Hearthstone. One way of evaluating our work is to compare our results with those achieved through existing work in the field.

2.2.1 Rule-Based

There are a handful of open-source Hearthstone simulators, including the one used in this project, SabberStone [33]. Many of these simulators include simple AI agents that work out of the box. These agents use basic heuristics to select moves that are predictable and not competitive.

2.2.2 Monte Carlo Tree Search

In addition to being the foundational algorithm of AlphaGo, Monte Carlo Tree Search (MCTS) is, by a wide margin, the most popular algorithm applied to Hearthstone. Playing Hearthstone with various boosted versions of MCTS have shown promising results. Some methods for boosting MCTS for Hearthstone include:

- Using domain specific knowledge [34],
- Advanced pruning techniques for effective rollout policies through bucketing similar chance events [35], and
- Applying a trained value network to MCTS for iterative network enhancement through self-play [7].

Each of these examples improve upon the performance of “vanilla” Monte Carlo for Hearthstone. However, none of these algorithms can beat a Legend ranked player more than 50% of the time. In all our research, the agent developed by Świechowski, Tajmajer, & Janusz has the highest winning percentage against Legend ranked players at 43% when going 2nd. However, when going 1st, that same agent wins only 17% of the time against Legend ranked players [7].

Therefore, the core motivation for this project was to use combinatorial fusion analysis to develop a combination model that outperforms each of the individual models that would have the potentional to be better than the existing Hearthstone playing AI agents.

2.3 Combination of Models using CFA

We used five machine learning models, which we list in *Section 3.3.2*, for our combiniatorial fusion analysis. After finding the score and rank functions for each of the five individual models, we find the score function of the score combination and the score function of the rank combination for all combinatorial combinations. This produces 31 different scoring systems, which we then evaluate and compare using the precision criterion as explained in *Section 1.3*.

2.4 Performance Evaluation

We use precision analysis to evaluate the performance of our models. Calculating the precision of a model is quite straightforward. First, we count the dataset's positive classifications. For Hearthstone, this is the number of games in which the player under analysis wins. Next, for each system, the games in the dataset are sorted by score in descending order, or by rank in ascending order. Finally, we can find the precision of the model by calculating the percentage of the top ranked games of the sorted dataset that have a positive classification. For our dataset of 500 games, this is the percentage of the top 291 games.

It is important to recognize that the precision of a model is different from its accuracy. A model's accuracy reflects its ability to correctly classify individual instances of a dataset. Precision, however, can only be determined once a model's score and rank functions have been defined and applied to each item in the dataset.

3 System Implementation

This chapter details the construction and implementation of our system, the methods used for data creation and processing, the strategy for applying CFA to Hearthstone. We also take the first section to describe the tools and environment that make up the backbone of the project.

3.1 Simulation Environment

Blizzard does not allow for computer AI agents to interact with the Hearthstone client. Therefore, we used HearthSim's SabberStone simulator, which is written in C#, for the construction of the dataset and all AI implementation, simulation, and testing [33]. We chose SabberStone for the following reasons:

- It is the most complete open-source simulator of those available (nearly all playable cards are implemented) .
- It is regularly updated and maintained.
- It comes with a straightforward framework for implementing and utilizing user-built AI agents.
- It is the simulator of choice for the annual Hearthstone AI Competition, which is in its third year, is sponsored and presented by the IEEE Conference on Games, and provides contestant's agents for download and use [36].

3.2 Data and Features

3.2.1 Dataset

Our dataset is comprised of 500 simulations between two separate instances of an existing MCTS AI agent. Each simulation is saved as a CSV with an action-by-action record of the key information available to both players at every game-state. These observations include information such as each player's health, the number of minions each player controls, the number of cards in each player's hand, etc.

3.2.1.1 Simulation Agents

We used Kai Bornemann's Tyche Agent to generate our dataset. Bornemann's agent uses MCTS and participated in the Hearthstone AI Competition in both 2018 and 2019, placing in the top 4 in both years in both the Premade and Custom Deck Playing Tracks. As explained in the previous section, the competition publishes contestant's agents for download and use [37].

3.2.1.2 Simulation Details

For each of the 500 simulations, we use two instances of Tyche Agent, one as the Paladin hero class and one as the Mage hero class, using the respective, basic decks shown in *Appendix C*. We use basic decks, decks that do not feature cards with complex abilities or effects, to limit the variability of each simulation. The idea is that a model built from the simplest scenario should be applicable to more complex scenarios. At the end of every simulation, we add the winner of the match as the class label.

3.2.2 Feature Set

We extracted twelve features from our dataset as the input for the machine learning models used in this project. In [38], Bursztein explored the relationship between five different metrics and predicting winning games of Hearthstone. We used these metrics as five of our twelve features because, like our system, Bursztein's model predicts a game's winner based on cumulative features observed throughout a match:

- **Mana Advantage** – The difference between the total mana each player has spent.
- **Board Mana Advantage** – The difference between the sum of the mana cost of both players' minions in play.

- **Board Count Advantage** – The difference in the count of minions in play for each player.
- **Draw Advantage –** The difference in the total number cards each player has drawn.
- **Hand Size Advantage** – The difference in the number of cards in each player's hand.

Bursztein's results show that each of these metrics have predictive power, with board mana advantage and mana advantage ranking as the top two features for classifying winning games. Furthermore, Bursztein's work inspired the rest of the features:

- **Available Attack Advantage** – The difference in the sum of attack values available to each player (including heroes).
- **Available Defense Advantage** – The difference in the sum of the health of taunt minions under each player's control.
- **Board Bonus Advantage** – The difference in the total bonus values of each player's board. These values were inspired by Bursztein's work on card appraisal [39].
- **Board Health Advantage** – The same as available defense advantage except the health of all minions is counted.
- **Board Ratio Advantage** – The difference in the sum of the ratios of each player's minions' attack to health.
- **Deck Count Advantage** – The difference in the number of cards remaining in each player's deck.
- **Health Advantage –** The difference in the players' total health remaining (armor is included as health points).

3.3 CFA with Hearthstone Data

3.3.1 Preprocessing the Data for the Scoring Systems

For each model, we derived a scoring system from the weight each feature contributes towards predicting the winner of a game. As previously explained, every simulation produces CSV file containing an action-by-action recording of all key observations throughout the game with the winner as the class label.

We built a Python program that extracts the feature set from each game and feeds them into the five models. While some of the systems described in *Section 2.2* predict the winner of a game when given a set of feature values from a single game-state, our system is designed to predict the winner of an entire game. Therefore, as mentioned previously, our preprocessing procedure extracts the cumulative values of each feature at the end of the game.

Practically speaking, we chose this method because we believe that learning from a game's cumulative results may solve the issues that arise when handling uncertainty. Issues that present a critical hinderance to MCTS Hearthstone stystems. In order to effectively apply MCTS to Hearthstone, a system must implement solutions to the problems associated with the game's partial information such as estimating the opponent's hand and deck for rollouts. By looking at a game in its entirety, our system assumes that the specific moves an opponent makes throughout the course of the game do not matter as much as how those moves contribute to the game's culmination.

3.3.2 Model Selection

CFA relies on the derivation of a scoring function from each individual model. These scoring functions are then used to compute a score for each instance in the dataset. Therefore, the simplest models to include in CFA are those that calculate feature weight or Gini importance.

Each score function finds the score of each game as the weighted sum of its features, see *Equation 3.1*. Where $\boldsymbol{M}$ is an individual model, $\boldsymbol{d_i}$ is a single game, $\boldsymbol{n}$ is the number of features in $\boldsymbol{d_i}$, $\boldsymbol{w_j}$ is the $\boldsymbol{j^{th}}$ weight in $\boldsymbol{M}$, and $\boldsymbol{g_j}$ is the $\boldsymbol{j^{th}}$ feature for all $\boldsymbol{i}$ games in the dataset.

$$s_M(d_i) = \sum_{\mathrm{j}=1}^{n=12} w_j * g_j(d_i),\ \forall\ i = 1,2, \dots ,500$$

Equation 3.1: Weighted Sum of Feature Values

With this criterion in mind, we chose the following five models from scikit learn for CFA of Hearthstone game data [40]:

- Liblinear SVM
- Linear Regression
- Decision Tree
- Random Forest
- AdaBoost

3.3.3 Combining the Models

Once each of the five models has been trained on the feature set using 5-fold cross validation, we compute the score of each of the 500 game simulations according to *Equation 3.1*. We then find the score of the score combination function and the score of the rank combination function, as described in *Section 1.3*, for all combinatorial combination of the five individual models.

4 Experimental Results and Discussion

In this chapter, we present the results of applying combinatorial fusion analysis to Hearthstone game data. After introducing the individual scoring systems and their results, we move onto the results of the best combinations. Next, we present the performance evaluation of every system and the effect of cognitive diversity between Hearthstone game scoring systems.

4.1 Scoring Systems

4.1.1 Individual Models: Feature Weights and Scoring Functions

As discussed in the previous chapter, the first step of CFA is to calculate the score and rank functions of the individual models. For readability, we refer to SVM as System A, Linear Regression as System B, Decision Tree as System C, Random Forest as System D, and AdaBoost as System E. The initial phase of training produced the feature weights plotted in *Figure 4.1* and shown in *Table 4.1*, and the subsequent score function tables.

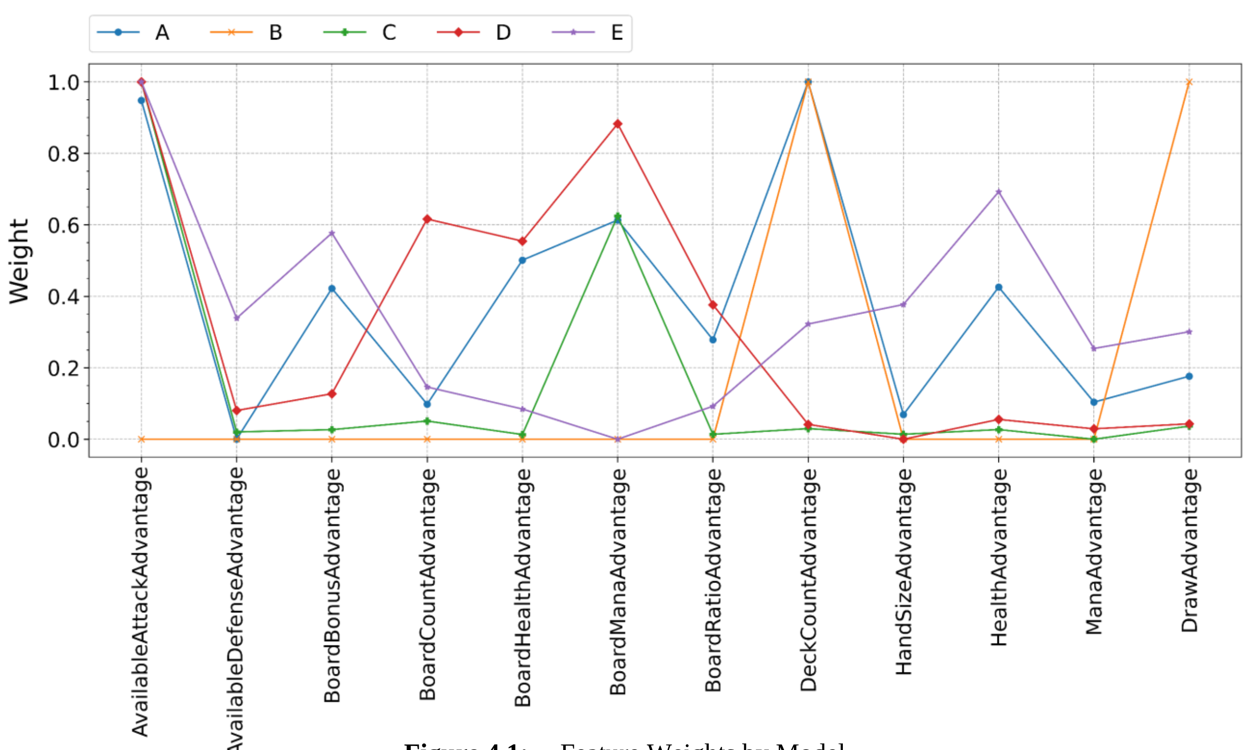

Figure 4.1: Feature Weights by Model

Feature	Weights				
	A	B	C	D	E
Available Attack Adv.	0.94801	0.00000	1.00000	1.00000	1.00000
Available Defense Adv.	0.00000	0.00000	0.01992	0.07942	0.33846
Board Bonus Adv.	0.42206	0.00000	0.02426	0.12958	0.57692
Board Count Adv.	0.09788	0.00000	0.05017	0.61542	0.14615
Board Health Adv.	0.50109	0.00000	0.01421	0.52599	0.08462
Board Mana Adv.	0.61316	0.00000	0.62883	0.88005	0.00000
Board Ratio Adv.	0.27837	0.00000	0.01352	0.37170	0.09231
Deck Count Adv.	1.00000	1.00000	0.03553	0.04179	0.33539
Hand Size Adv.	0.06882	0.00000	0.01029	0.00000	0.37692
Health Adv.	0.42568	0.00000	0.02650	0.05565	0.69231
Mana Adv.	0.10376	0.00000	0.00000	0.02765	0.25385
Draw Adv.	0.17627	1.00000	0.03375	0.04201	0.28769

Table 4.1: Feature Weights by Model

Rank	f_A	Game	W/L
0	1	255	1
1	0.922715	471	1
2	0.90884	286	1
3	0.894606	455	1
⋮	⋮	⋮	⋮
277	0.543699	29	1
278	0.542092	424	1
279	0.536899	136	1
280	0.535842	150	1
281	0.534	75	1
282	0.520266	499	1
⋮	⋮	⋮	⋮
495	0.0806049	153	0
496	0.0581929	443	0
497	0.036906	466	0
498	0.0237644	320	0
499	0	354	0

Table 4.2: RSC Function – System A

Rank	f_B	Game	W/L
0	1	255	1
1	0.936481	122	1
2	0.916306	333	1
3	0.894875	162	1
⋮	⋮	⋮	⋮
277	0.551098	172	1
278	0.551071	266	1
279	0.544922	499	1
280	0.543442	29	1
281	0.541668	32	1
282	0.539524	387	0
⋮	⋮	⋮	⋮
495	0.0758743	354	0
496	0.0671501	320	0
497	0.0253601	8	0
498	0.0141678	466	0
499	0	153	0

Table 4.3: RSC Function – System B

Rank	f_C	Game	W/L
0	1	255	1
1	0.85098	426	1
2	0.833833	122	1
3	0.818835	73	1
⋮	⋮	⋮	⋮
277	0.477537	199	1
278	0.475749	233	1
279	0.470163	251	1
280	0.465393	307	1
281	0.46514	62	0
282	0.458611	268	1
⋮	⋮	⋮	⋮
495	0.0550915	208	0
496	0.0227144	320	0
497	0.00592327	153	0
498	0.00301707	8	0
499	0	354	0

Table 4.4: RSC Function – System C

Rank	f_D	Game	W/L
0	1	255	1
1	0.893895	122	1
2	0.850151	333	1
3	0.849531	286	1
⋮	⋮	⋮	⋮
277	0.518855	29	1
278	0.5181	251	1
279	0.517164	22	1
280	0.51658	199	1
281	0.511193	56	0
282	0.510378	62	0
⋮	⋮	⋮	⋮
495	0.0762874	466	0
496	0.040353	354	0
497	0.0355949	320	0
498	0.00817378	8	0
499	0	153	0

Table 4.5: RSC Function – System D

Rank	f_E	Game	W/L
0	1	255	1
1	0.984921	122	1
2	0.975777	162	1
3	0.971502	333	1
⋮	⋮	⋮	⋮
277	0.594341	229	1
278	0.593277	465	1
279	0.593221	205	1
280	0.592282	424	1
281	0.590152	268	1
282	0.585165	136	1
⋮	⋮	⋮	⋮
495	0.140535	432	0
496	0.103536	320	0
497	0.0767977	153	0
498	0.0523064	354	0
499	0	466	0

Table 4.6: RSC Function – System E

4.1.2 Score of SC and Score of RC Results for Top Combinations

From the score and rank functions of the individual models we can derive the same functions for other combinations. Here, we include the RSC Function of both the score and rank combination tables for the top combinations of ABE and ABDE, respectively. For the score and rank combination tables not included here, see *Appendix D*.

Rank	s(SC(ABE))	Game	W/L
0	1	255	1
1	0.928279	122	1
2	0.925506	333	1
3	0.910351	162	1
⋮	⋮	⋮	⋮
277	0.561438	199	1
278	0.555339	387	0
279	0.554036	266	1
280	0.550088	136	1
281	0.548401	465	1
282	0.543436	499	1
⋮	⋮	⋮	⋮
495	0.106795	8	0
496	0.0648168	320	0
497	0.0524675	153	0
498	0.0427269	354	0
499	0.0170246	466	0

Table 4.19.a: RSC Function of the Score Combination – System ABE

Rank	s(RC(ABE))	Game	W/L
0	0	255	1
1	2.33333	122	1
2	3	333	1
3	3.66667	286	1
⋮	⋮	⋮	⋮
277	275	29	1
278	277.667	387	0
279	279.667	266	1
280	282.667	465	1
281	282.667	499	1
282	284	136	1
⋮	⋮	⋮	⋮
495	492	249	0
496	496.667	320	0
497	497	153	0
498	497.333	354	0
499	498	466	0

Table 4.19.b: RSC Function of the Rank Combination – System ABE

Rank	s(SC(ABDE))	Game	W/L
0	1	255	1
1	0.919683	122	1
2	0.906667	333	1
3	0.894114	162	1
⋮	⋮	⋮	⋮
277	0.550223	199	1
278	0.546593	387	0
279	0.54621	465	1
280	0.542886	136	1
281	0.541672	499	1
282	0.541318	266	1
⋮	⋮	⋮	⋮
495	0.0821398	8	0
496	0.0575113	320	0
497	0.0421335	354	0
498	0.0393506	153	0
499	0.0318403	466	0

Table 4.29.a: RSC Function of the Score Combination – System ABDE

Rank	s(RC(ABDE))	Game	W/L
0	0	255	1
1	2	122	1
2	2.75	333	1
3	3.5	286	1
⋮	⋮	⋮	⋮
277	275.75	199	1
278	277.25	387	0
279	279.25	465	1
280	279.75	499	1
281	281	266	1
282	281.75	136	1
⋮	⋮	⋮	⋮
495	493.25	8	0
496	496.75	320	0
497	497	354	0
498	497.25	466	0
499	497.5	153	0

Table 4.29.b: RSC Function of the Rank Combination – System ABDE

4.2 Cognitive Diversity in Hearthstone Game Scoring Systems

4.2.1 Rank-Score Characteristic (RSC) Function

According to Hsu and Taksa [22], we can predict the rank combinations of two models that are most likely to have better performance than the corresponding score combination *prior* to calculating the precision of each combination's score and rank functions. To make such a prediction, we look at the Rank-Score Characteristic (RSC) function graph, which plots the score of each game vs. its rank. Under certain conditions, with the primary condition being high cognitive diversity, when combining two models with high cognitive diversity, we can expect that combination to have a more precise rank combination than score combination [22]. In this case, we should especially expect this result when combining systems C and E as indicated in *Figure 4.2*, *Figure 4.3*, and *Table 4.33.* Additionally, *Figure 4.4* visualizes the cognitive diversity between each pair of the 5 individual scoring systems.

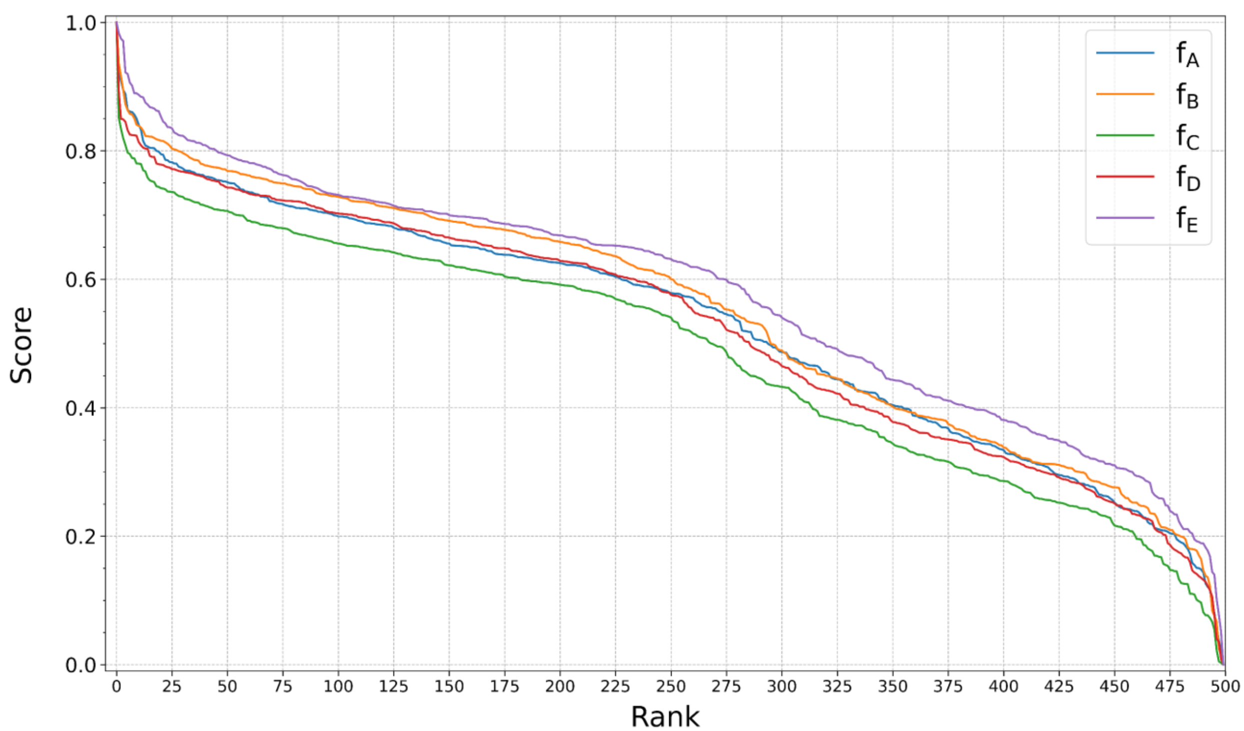

Figure 4.2: Rank-Score Characteristic (RSC) Function Graphs f_A, f_B, f_C, f_D, and f_E for the Five Models A, B, C, D, and E, respectively for 500 Game Simulations

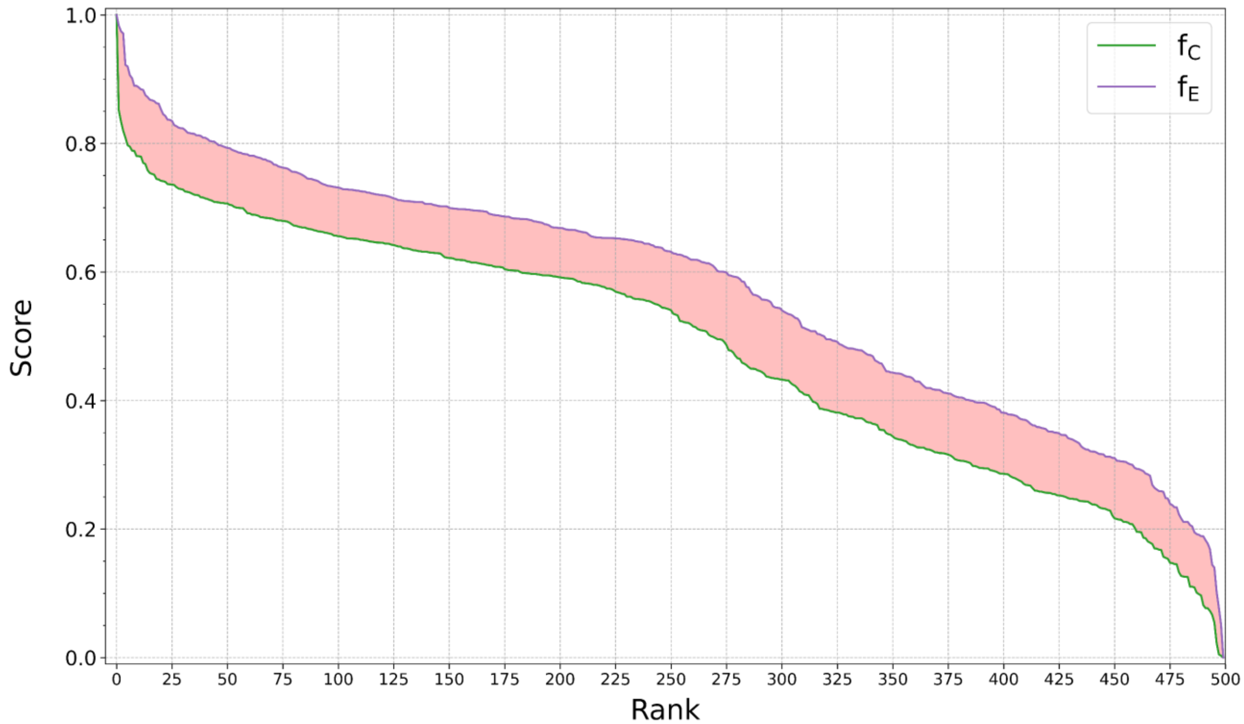

Figure 4.3: Cognitive Diversity Between System C and System E Illustrated

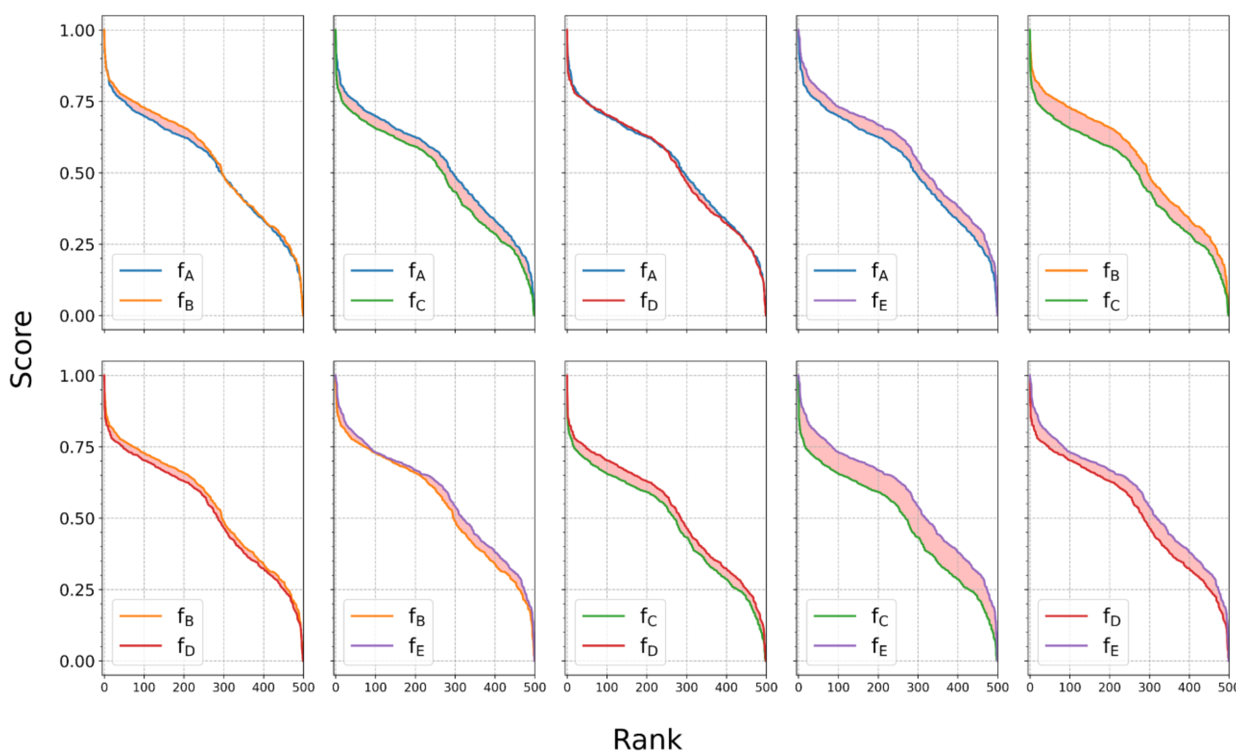

Figure 4.4: Cognitive Diversity Between All 10 Pairs of 5 Individual Models Illustrated

Model Pair	CD
CE	2.085976
BC	1.427536
DE	1.235623
AE	1.060047
AC	1.046779
CD	0.886772
BE	0.742508
BD	0.556233
AB	0.497215
AD	0.313584

Table 4.33: Cognitive Diversity (CD) of Model Pairs

4.3 Performance Comparison

Finally, once all score scombination and rank combination functions have been defined for all model combinations, we solve for the precision of each and plot them together as shown below in *Figure 4.5*.

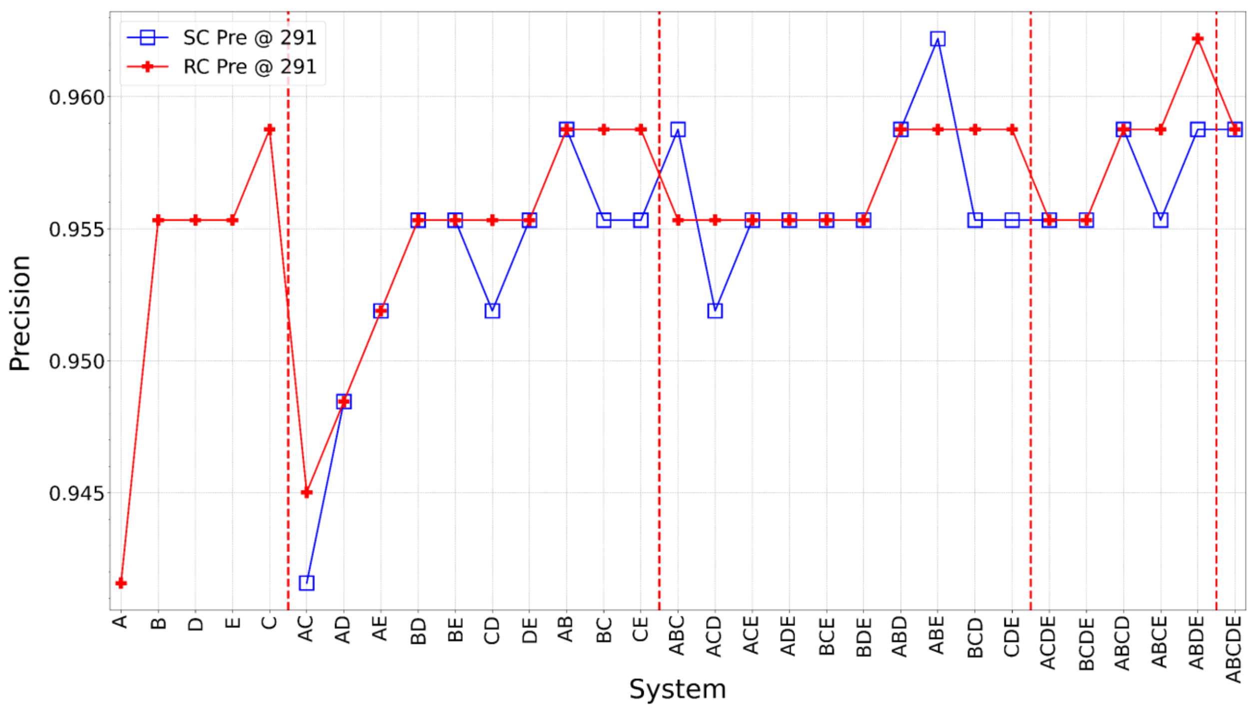

Figure 4.5: Score Combination vs. Rank Combination - Pre @ 291

4.3.1 Positive/Negative Precision Improvement

According to [21, 23], the performance of the combination of two models *X* and *Y* is related to the cognitive diversity between, and the performance ratio of, *X* and *Y*. It is a positive case if the performance of the SC or RC combination is better than the best performing of the two systems *X* and *Y*. Otherwise, it is a negative case.

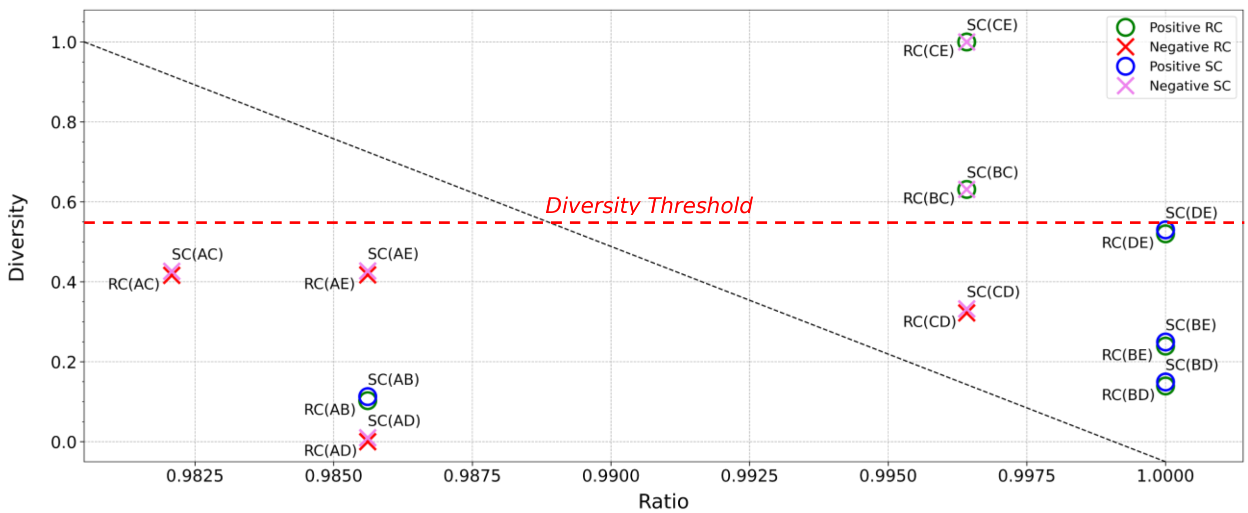

Figure 4.6: Positive/Negative Cases for 2 Combinations - 20 Combinations (10 SC and 10 RC of two models)

System	Ratio	Diversity
RC(AB)	0.985612	0.103607
RC(BC)	0.996416	0.628502
RC(BD)	1.0	0.136905
RC(BE)	1.0	0.242003
RC(CE)	0.996416	1.0
RC(DE)	1.0	0.520223

Table 4.34.a: Positive Rank Combination Cases of 20 Combinations

System	Ratio	Diversity
SC(AB)	0.985612	0.103607
SC(BD)	1.0	0.136905
SC(BE)	1.0	0.242003
SC(DE)	1.0	0.520223

Table 4.34.b: Positive Score Combination Cases of 20 Combinations

System	Ratio	Diversity
RC(AC)	0.982079	0.413676
RC(AD)	0.985612	0.0
RC(AE)	0.985612	0.421162
RC(CD)	0.996416	0.323398

Table 4.35.b: Negative Rank Combination Cases of 20 Combinations

System	Ratio	Diversity
SC(AC)	0.982079	0.413676
SC(AD)	0.985612	0.0
SC(AE)	0.985612	0.421162
SC(BC)	0.996416	0.628502
SC(CD)	0.996416	0.323398
SC(CE)	0.996416	1.0

Table 4.35.b: Negative Score Combination Cases of 20 Combinations

4.4 Discussion

We can make three definitive conclusions from these results. First, the performance comparison shown in *Figure 4.5* confirms our prediction that the rank combination of systems C and E would outperform the score combination of C and E. We also see similar results in the combinations of systems A and C, C and D, and B and C. Furthermore, the results illustrated in *Figure 4.6* and its corresponding tables demonstrate that in this dataset, the combinations of models C and E and B and C seem to define a "diversity threshold" of 0.55. When a combination of two scoring systems crosses that threshold, the rank combination performs at least as well as the combination's best performing component, while the score combination performs worse than its best performing component.

Secondly, *Figure 4.6* and the corresponding tables highlight that the performance when combining two models with similar performance and high diversity will improve upon the performance of the combination's individual models. This is true for all models except for the combinations of A and B and C and D. Additionally, if we look at the positive/negative cases for all the two combinations that can be made from the best overall systems, the score combination of ABE and the rank combination of ABDE, all but AD and AE show a positive case with diversity below the threshold.

Further analysis of *Figure 4.5* reveals more insight into the relationship between individual models used in this project. First, *Figure 4.5* shows that the combination of all models, ABCDE does not improve in performance over the best individual model. This should be expected given the low performance of most combinations of two models that include system A. Next, looking at the best performing models: the score combination of ABE and the rank combination of ABDE, we observe that both combinations outperform the best individual model, C, without including C. This result demonstrates one of the key aspects of model diversity in this environment: that it is not the best individual that contributes to the best combination; rather it is the combination that prioritizes the best performing and most diverse individual models.

Lastly, the score combination of ABE and the rank combination of ABDE outperform their most precise component, and are therefore the best candidates for a Hearthstone AI agent's game-state scoring function and system refinement. As a final experiment, we refined the system by retraining the models with the top ranked games of SC(ABE), and then performed CFA on the retrained models. Prior to retraining we expected that some two-combination of A, B, and E would show even greater performance improvement than the original system. This prediciton was confirmed, as depicted in *Figure 4.7*, with the new score combination of systems B and E achieving a precision of over 98%. While this is possibly a result of overfitting, it is worth presenting as it followed our expectations and demonstrates a performance enhancement trend through iterative CFA.

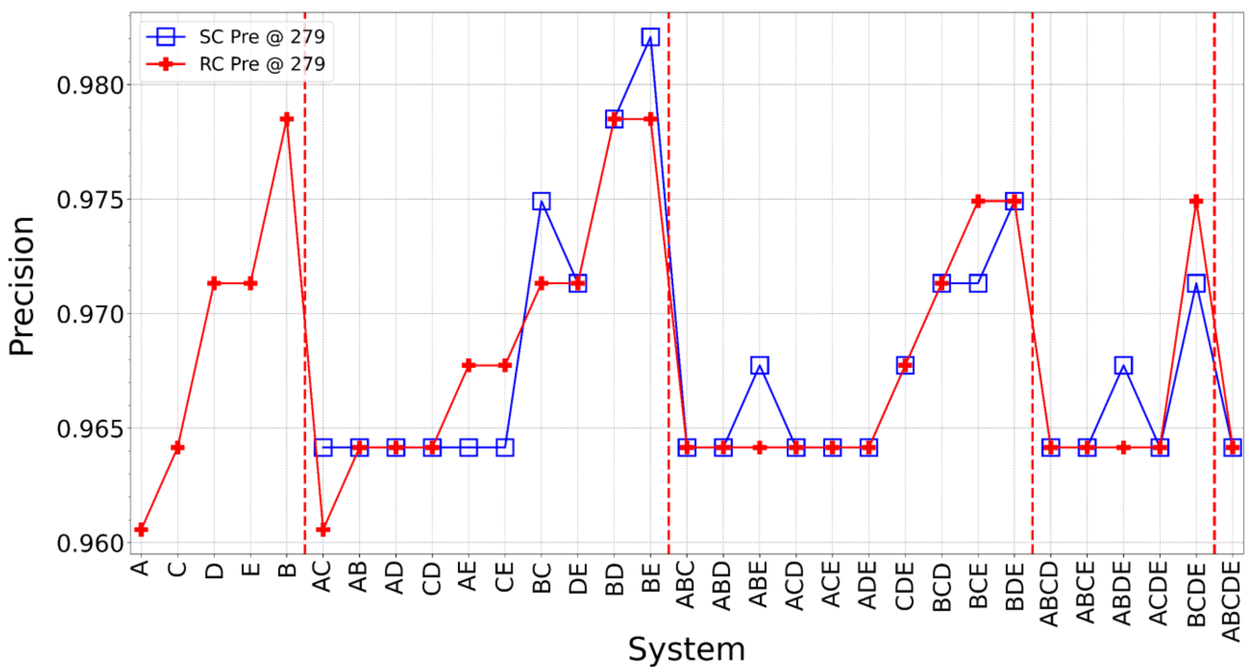

Figure 4.5: Performance of All Systems – Retrained with Top 291 Ranked Games of Original SC(ABE)

5 Summary and Further Work

5.1 Summary

Our application of combinatorial fusion analysis to Hearthstone game data is very promising. We proved that under certain conditions, the presence of cognitive diversity in Hearthstone scoring systems will produce a fused model that outperforms its individual components. Additionally, we demonstrated the power of CFA as an empirical performance comparison method for both model fusion and model selection with Hearthstone game-state data. Lastly, these data-driven results indicate that under the experimental conditions used to generate our dataset, a Hearthstone playing AI agent using the ABE score combination, the ABDE rank combination, or the retrained BE score combination scoring system should win over 96-98 times in 100 games.

5.2 Issues

That said, our existing dataset places limitations on the system in its current state that we could solve in future work.

5.2.1 Small Dataset

Many of the previous projects described in *Section 2.2.2* use datasets with thousands, or tens of thousands of games. The authors of [7] built their own simulator that could reach speeds of 30k games per second, which obliterates our SabberStone environment's rate of 500 games in about six hours. So, although 5-fold cross validation was used in training the individual models, the high performance suggests possible overfitting.

5.2.2 Narrow Dataset

Our dataset is extremely narrow for three primary reasons:

- It only includes games between two of Hearthstone's nine hero classes, and we only perform analysis on one of those heroes. Even a casual Hearthstone player recognizes the nuances of playing with each hero class, and how class abilities and synergies affect a player's strategy.
- We only used one deck each for the hero classes we tested. Hero class, deck, strategy, and opponent reciprocally influence one another. By holding hero class, deck, and opponent constant, our system learns the best strategy when playing with a specific deck against a specific opponent. And it is therefore unlikely that our system would fit if we changed one of those constants without retraining the models.
- We used one bot—albeit with parameters set specifically for each hero class—to construct our dataset. This runs the risk of our model only learning how to beat a specific AI opponent, which might not be superior to other Hearthstone playing AIs, let alone a human opponent.

5.3 Possible Future Work

5.3.1 Ideal Dataset

While these issues must be acknowledged, the results we have presented in this work are both valid and valuable. Our system is inherently scalable as it is built from scalable machine learning models. Therefore, with an ideal dataset, we have demonstrated that CFA alone could potentially produce a versatile and superior Hearthstone playing AI agent. The ideal dataset would need to have the following characteristics:

- **Large number of games:** As the size of the dataset increases, so should the performance of the system.

- **Real, competitive games:** A dataset including real game data between players in the top 18.3% of Hearthstone's player rankings would be an excellent starting point for developing an AI agent with superhuman skill [41].
- **Comprehensive and diverse:** Such a dataset would need to include games between each combinatorial combination of the 9 hero classes playing with a wide variety of decks. This would allow for further development of our system so that it would learn the general, optimal playing strategy for each hero class against each opponent, which in turn could be honed for specific decks.

5.3.2 Boosting Monte Carlo

In addition to—or instead of—a dataset that is even close to this ideal, we have demonstrated that we could use our CFA methods to boost Monte Carlo Tree Search for Hearthstone. As previously explained, one of the primary reasons MCTS is the most popular algorithm used for Hearthstone, and overall game playing, is that it is designed to mitigate uncertainty and capitalize on probability. However, where MCTS falls short is that its success relies on the development of a strong mitigation strategy. Therefore, our CFA methods could boost MCTS in two ways:

- To choose and refine the best MCTS strategy, possibly using SC(ABE), RC(ABDE), or the retrained SC(BE) as the exploitation factor of the Upper Confidence bound applied to Trees formula, and/or
- To combine multiple strategies and/or a different set of initial models.

Such an algorithm could potentially produce a Hearthstone playing AI agent capable of competing with the best existing individual models and possibly with the best human players.

6 References

[1] R. Munos, "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning," *Foundations and Trends® in Machine Learning,* vol. 7, no. 1, pp. 1-129, 2014.

[2] D. Lee, "Game theory and neural basis of social decision making," *Nature Neuroscience,* vol. 11, no. 4, pp. 404-409, 2008.

[3] A. Kurenkov, "A 'Brief' History of Game AI Up To AlphaGo," 18 April 2016. [Online]. Available: https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai/. [Accessed 3 March 2020].

[4] C. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine,* vol. 41, no. 314, pp. 256-275, March 1950.

[5] A. Kurenkov, "A 'Brief' History of Game AI Up To AlphaGo, Part 2," 18 April 2016. [Online]. Available: https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai-part-2/. [Accessed 3 March 2020].

[6] A. Kurenkov, "A 'Brief' History of Game AI Up To AlphaGo, Part 3," 6 April 2016. [Online]. Available: https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai-part-3/. [Accessed 3 March 2020].

[7] M. Świechowski, T. Tajmajer and A. Janusz, "Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms," *2018 IEEE Conference on Computational Intelligence and Games (CIG),* pp. 1-8, 2018.

[8] Hearthstone Wiki, "Advanced Rulebook," 23 December 2019. [Online]. Available: https://hearthstone.gamepedia.com/Advanced_rulebook. [Accessed 23 February 2020].

[9] M. H. Andersson and H. H. Hesselberg, Programming a Hearthstone agent using Monte Carlo Tree Search, Trondheim, 2016.

[10] Hearthstone Wiki, "Card Set," 7 April 2020. [Online]. Available: https://hearthstone.gamepedia.com/Card_set.

[11] S. Staff, "Dealing with data. Challenges and opportunities. Introduction.," *Science,* vol. 331, no. 6018, pp. 692-3, 2011.

[12] R. E. Schapire and Y. Freund, Boosting: Foundations and Algorithms, MIT Press, 2012.

[13] V. Vapnik, The Nature of Statistical Learning Theory, (2nd Edition), Springer, 2000.

[14] L. Brieman, "Random Forests," *Machine Learning,* vol. 45, pp. 5-32, 2001.

[15] C. M. Ginn, P. Willett and J. Bradshaw, "Combination of molecular similarity measures using data fusion," *Perspectives in Drug Discovery and Design,* vol. 20, no. 1, pp. 1-16, 2000.

[16] L. I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, 2nd Edition, Wiley, 2014.

[17] A. J. Sharkey, Ed., Combining Artificial Neural Nets, Springer, 1999.

[18] S. Wu, Data Fusion in Information Retrieval, Springer, 2012.

[19] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, Cambridge: CRC Press, 2012.

[20] D. F. Hsu, Y.-S. Chung and B. S. Kristal, "Combinatorial Fusion Analysis: Methods and Practices of Combining Multiple Scoring Systems," in *Advanced Data Mining Technologies in Bioinformatics,* H. Hsu, Ed., Hershey, Pennsylvania: Idea Group Publishing, 2006, pp. 32-62.

[21] D. F. Hsu, B. S. Kristal and C. Schweikert, *Rank-Score Characteristics (RSC) Function and Cognitive Diversity,* vol. 6334, Y. Yao, R. Sun, T. Poggio, J. Liu, N. Zhong and J. Huang, Eds., Berlin, Heidelberg: Springer-Verlag, 2010, pp. 42-54.

[22] D. F. Hsu and I. Taksa, "Comparing Rank and Score Combination Methods for Data Fusion in Information Retrieval," *Information Retrieval,* vol. 8, pp. 449-480, 2005.

[23] D. F. Hsu, B. S. Kristal, Y. Hao and C. Schweikert, "Cognitive Diversity: A Measurement of Dissimilarity Between Multiple Scoring Systems," *Journal of Interconnection Networks,* vol. 19, no. 1, pp. 1940001:1-1940001:42, 2019.

[24] D. M. Lyons and D. F. Hsu, " Combining multiple scoring systems for target tracking using rank-score characteristics," *Information Fusion,* vol. 10, no. 2, pp. 124-136, 2009.

[25] J.-M. Yang, Y.-F. Chen, T.-W. Shen, B. S. Kristal and D. F. Hsu, "Consensus Scoring Criteria for Improving Enrichment in Virtual Screening," *J. Chem. Inform. Model,* vol. 45, pp. 1134-1146, 2005.

[26] C. Schweikert, S. Shimojo and D. F. Hsu, "Detecting preferences based on eye movement using combinatorial fusion," in *2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, Palo Alto, 2016.

[27] C. Schweikert, S. M. Brown, Z. Tang, P. R. Smith and D. F. Hsu, "Combining multiple ChIP-seq peak detection systems using combinatorial fusion," *BMC Genomics,* vol. 13, p. S12, 2012.

[28] H. D. Vinod, D. F. Hsu and Y. Tian, "Combinatorial Fusion for Improving Portfolio Performance," in *Advances in Social Science Research Using R*, Heidelberg: Springer, 2010, pp. 95-105.

[29] P. Willett, "Combination of Similarity Rankings Using Data Fusion," *J. Chem. Inf. Model,* vol. 53, no. 1, pp. 1-10, 2013.

[30] D. F. Hsu, J. Shapiro and I. Taksa, "Methods of Data Fusion in Information Retreival: Rank vs. Score Combination," DIMACS TR 2002-58, 2002.

[31] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman and D. Grewe, "Mastering the game of Go with deep neural networks and tree search," *Nature,* no. 529, pp. 484-489, 2016.

[32] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 500, pp. 354-359, 2017.

[33] HearthSim, "SabberStone," 31 December 2019. [Online]. Available: https://github.com/HearthSim/SabberStone. [Accessed 31 December 2019].

[34] A. Santos, P. A. Santos and F. S. Melo, "Monte Carlo Tree Search Experiments in Hearthstone," *2017 IEEE Conference on Computational Intelligence and Games (CIG),* pp. 272-279, 2017.

[35] S. Zhang and M. Buro, "Improving hearthstone AI by learning high-level rollout policies and bucketing chance node events," *2017 IEEE Conference on Computational Intelligence and Games (CIG),* pp. 309-316, 2017.

[36] A. Dockhorn and S. Mostaghim, "Introducing the Hearthstone-AI Competition," *ArXiv, abs/1906.04238.,* 2019.

[37] A. Dockhorn and S. Mostaghim, "Bot Downloads," 2019. [Online]. Available: https://dockhorn.antares.uberspace.de/wordpress/bot-downloads/.

[38] E. Bursztein, "I am a legend: hacking hearthstone using statistical learning methods," *2016 IEEE Conference on Computational Intelligence and Games (CIG),* pp. 1-8, 2016.

[39] E. Bursztein, "How to appraise Hearthstone card values," July 2014. [Online]. Available: https://elie.net/blog/hearthstone/how-to-appraise-hearthstone-card-values/.

[40] F. Pendregosa, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[41] A. D. A. (@IksarHS), *Rank distribution for the end of November 2019...,* Twitter, 2020.

[42] Hearthstone Wiki, "Gameplay," 7 July 2019. [Online]. Available: https://hearthstone.gamepedia.com/Gameplay. [Accessed 2019 10 9].

[43] Hearthstone Wiki, "Card," 6 April 2020. [Online]. Available: https://hearthstone.gamepedia.com/Card.

Appendix A – Hearthstone: A Beginner's Guide

This Appendix provides a complete overview of Hearthstone, its rules, and its mechanics. In order to understand our system and how combinatorial fusion analysis can be applied to Hearthstone, readers must understand the game's core concepts. While the density of the following sections may be intimidating, in practice, and as previously stated, this game is easy to learn, but hard to master. Therefore, we suggest downloading Hearthstone to your computer or mobile device and playing for a short period to supplement or replace this section.

A.1. Gameplay Summary

Adapted from [42] *and* [9]

Each Hearthstone match is played as a one-on-one duel between two opponents. Gameplay in Hearthstone is turn-based, with players alternating playing cards from their hand to cast *spells*, equip *weapons*, or summon *minions* to do battle on their behalf.

Players are represented by their selected *hero*. Each hero is associated with a *hero class*, which specifies a unique set of cards available for *deck* construction, and a unique *hero power* that can be used during gameplay. At the start of a duel, a hero has 30 health points. A hero's health is always displayed in the blood drop on the character portrait as shown in *Figure A.1*. When a hero's health is reduced to zero, the controlling player loses. Thus the goal of a Hearthstone game is straightforward—although difficult to achieve—reduce your enemy's health to zero.

Figure A.1: Hearthstone hero Uther Lightbringer, Paladin Hero Class

At the start of each turn, the current player draws the top card from their *deck*: a collection of 30 cards assembled and then shuffled before play begins. Players can choose to play using one of several pre-assembled decks, or to construct and play with a custom deck. Most cards are Neutral, meaning that they can be included in any deck. However, as explained previously, a substantial portion of cards are limited to a specific class, giving each hero their own strengths and unique abilities.

During a player's turn, that player attempt to play any of their cards, use their hero power, use minions to attack targets, or use their hero to attack directly if they have a weapon equipped or an attack value. Most actions, however, such as playing cards, require the player to spend mana crystals. Mana is the limiting resource that forces players to strategically plan out their moves. Players starts the game with 1 mana crystal and gains one additional mana crystal at the start of their turn up to the maximum of 10 mana crystals. Furthermore, at the start of a player's turn, their mana crystals are fully restored. Any unspent mana remaining at the end of the turn does not carry over to the next. So, for example, Player A starts his turn with five mana crystals. On his turn, he uses four mana. On his next turn, his five mana crystals are restored, and he gains one additional mana crystal for a total of six to be used on his turn. It should be clear that as the game progresses, players have access to more mana crystals, which allows them to play more cards per turn, or cards with higher costs.

Hearthstone features multiple strategic elements that players must master to play competitively. The positioning and control of minions, assigning accurate strategic importance to various variables, capitalizing on complex card synergies and interactions, and working around a shuffled deck are some of aspects that combine to make Hearthstone an intricate game.

A.2. Battlefield

The game takes place on a game board called the battlefield. *Figure A.2* shows an example. The battlefield is the user interface for a game of Hearthstone and contains all important elements required to play such as the player's hand, deck, mana crystals, minions, and

Figure A.2: Battlefield

hero. The battlefield is always shown from your perspective with your minions, hero, hand, deck, and mana crystals all located in the bottom half. All elements are mirrored for the opponent in the top half of the battlefield, but your opponent sees everything from their perspective (in the bottom half).

A.3. Order of Play

A.3.1. Start of Game

Before the game begins, a coin toss decides which player goes first. Players are then shown cards randomly drawn from their own deck (three cards for the player going first, four cards for the player going second). Players then choose to keep or replace these cards

Figure A.3: Mulligan

individually. This is called the mulligan (*Figure A.3*). Cards are replaced randomly from a player's deck, and the replaced cards are shuffled back into the player's deck. In addition to starting with an extra card, the player going second also receives the coin from the coin toss as a special card. "The Coin" can be used at any time granting the player one extra full mana crystal until the end of the turn. These two extra cards serve to offset the inherent disadvantage of going second.

A.3.2. On Your Turn

At the start of each turn, players draw the top card from their deck, their mana crystals are refreshed, and they gain one additional mana crystal (up to the maximum of ten). Additionally, a player can have a maximum of ten cards in their hand. Any cards drawn with a full hand are revealed to both players, and then destroyed.

During your turn you can play any of your cards, such as minions, spells, and weapons, provided you have enough mana. Playing a card will consume the amount of mana indicated at the top-left corner of the card. You are also able to command your minions to attack, use your hero's hero power, or use your hero to attack targets directly if your hero has an attack value (usually granted by having a weapon equipped, but can be achieved in other ways).

Players have 75 seconds to complete their turn but can elect to end their turn at any point before the time limit. After 75 seconds, the turn automatically ends.

If a player attempts to draw cards when there are none remaining in their deck, they will take *fatigue* damage. Fatigue damage starts at one and increments by one every time a player attempts to draw cards from an empty deck.

A.3.3. Conclusion of a Match

A duel ends when one of the following conditions is met:

Figure A.4: The Four Horsemen Hero Power

- One of the players' heroes reaches zero health (or below) and is destroyed. The remaining player is the victor.
- One of the players achieves a victory condition specified by a card or hero power (see *Figure A.4*).
- A player concedes (see below) or leaves the game, the other player wins.
- Both heroes' health reaches zero at the same time, the game ends in a draw.
- The game reaches the 90th turn, the game ends in a draw.

Appendix B – Calculating the Number of Possible Moves on a Turn

B.1. Number of Possible Attacking Moves

The number of possible attacking moves player x can execute is equal to A_x and is defined as follows:

$$A_x = {C_y}^{C_x}$$

Equation B.1.1: Number of Possible Attacking Moves

Where C_y is the number of characters controlled by opponent y that can be attacked, and C_x is the number of characters controlled by player x that can attack. A character can attack if and only if it has an attack value ≥ 0, it is not exhausted, and it is not frozen. C_y is found according to the following filter:

$$C_y = \begin{cases} \text{the number of taunt minions} & y \text{ controls taunt minions} \\ \text{All characters controlled by } y \text{ that do not have stealth} & \text{otherwise} \end{cases}$$

Equation B.1.2: Filter for Attackable Minions

B.1.1. Maximum Value of A_x

In practice, the scenario that produces the maximum value of $A_x = 8^8$ is not uncommon and has the following characteristics:

- Both players control 7 minions
- The attacking player's hero has an attack value greater than zero
- The opponent does not have any stealth or taunt minions

B.2. Number of Possible Orders of Attack

The number of possible orders in which player x can execute A_x attacks is equal to the total permutations of the T number of attacks player m can execute. T is equal to the sum of the number of attacks each character controlled by player x that can attack can make.

B.2.1. Maximum Value of T

In a real game, the probability of seeing the scenario that produces the maximum value of $T = 30$ is incredibly small. Nonetheless, such a scenario exists and has the following characteristics:

- The attacking player controls “Whirlwind Tempest” which gives minions with windfury mega-windfury (allowing them to attack 4 times instead of twice)
- The attacking player has given “Whirlwind Tempest” windfury
- The attacking player controls 6 windfury minions in addition to “Whirlwind Tempest”
- The attacking player’s hero has “Doomhammer” equipped, thus granting the hero windfury
- Only the last of the 30 attacks can be lethal (the attacking player cannot win with prior to the 30^{th} attack)

B.3. Number of Targets, Playable Cards, and Order of Playable Cards

B.3.1. Maximum Number of Targetable Characters

The number of targetable characters M is equal to the total number of characters that do not have ***cannot be targeted by spells or hero powers*** or ***stealth*** abilities. Like A_x, a scenario that produces the maximum value of $M = 16$ is not uncommon and has the following characteristics:

- Both players control 7 minions
- Neither hero, nor any of the minions in play have ***cannot be targeted by spells or hero powers*** or ***stealth*** abilities

B.3.2. Maximum Number of Playable Cards

The number of playable cards P is simply the number of cards a player can afford to play according to the cost of each card and the player’s available mana crystals. There are many scenarios that could produce the value of $P = +\infty$. But, for the sake of useful mathematical expressions, we use the maximum value of $P = 10$, which is also not uncommon and has the following characteristics: the current player has 10 mana crystals, holds 10 spells in their hand that each cost 1 mana and neither require a target nor have conditional playability.

Appendix C – Decks Used for Dataset Construction

Figure C.1.a: Basic Paladin Deck

Figure C.1.b: Basic Mage Deck

Appendix D – Score and Rank Combination Tables

Rank	s(SC(AB))	Game	W/L
0	1	255	1
1	0.902508	333	1
2	0.899958	122	1
3	0.886007	286	1
⋮	⋮	⋮	⋮
277	0.54357	29	1
278	0.533275	197	1
279	0.532594	499	1
280	0.532549	136	1
281	0.532078	199	1
282	0.529764	232	1
⋮	⋮	⋮	⋮
495	0.0573492	8	0
496	0.0454573	320	0
497	0.0403025	153	0
498	0.0379372	354	0
499	0.0255369	466	0

Table 4.7.a: RSC Function of the Score Combination - System AB

Rank	s(RC(AB))	Game	W/L
0	0	255	1
1	3	333	1
2	3	122	1
3	3.5	286	1
⋮	⋮	⋮	⋮
277	278.5	29	1
278	280	197	1
279	280.5	499	1
280	282	199	1
281	283	232	1
282	284.5	266	1
⋮	⋮	⋮	⋮
495	495.5	8	0
496	497	354	0
497	497	153	0
498	497	320	0
499	497.5	466	0

Table 4.7.b: RSC Function of the Rank Combination - System AB

Rank	s(SC(AC))	Game	W/L
0	1	255	1
1	0.853377	286	1
2	0.848633	122	1
3	0.845192	455	1
⋮	⋮	⋮	⋮
277	0.506977	69	0
278	0.506696	117	1
279	0.505509	307	1
280	0.502905	197	1
281	0.499275	281	0
282	0.494425	30	0
⋮	⋮	⋮	⋮
495	0.0591909	466	0
496	0.0461776	8	0
497	0.0432641	153	0
498	0.0232394	320	0
499	0	354	0

Table 4.8.a: RSC Function of the Score Combination - System AC

Rank	s(RC(AC))	Game	W/L
0	0	255	1
1	3.5	286	1
2	3.5	122	1
3	4.5	455	1
⋮	⋮	⋮	⋮
277	277.5	307	1
278	278	75	1
279	278.5	197	1
280	278.5	117	1
281	281	281	0
282	281	30	0
⋮	⋮	⋮	⋮
495	494	443	0
496	496	153	0
497	496	8	0
498	497	320	0
499	499	354	0

Table 4.8.b: RSC Function of the Rank Combination - System AC

Rank	s(SC(AD))	Game	W/L
0	1	255	1
1	0.879185	286	1
2	0.878664	122	1
3	0.869431	333	1
⋮	⋮	⋮	⋮
277	0.528322	499	1
278	0.527786	117	1
279	0.527221	32	1
280	0.524425	69	0
281	0.5228	421	0
282	0.519469	281	0
⋮	⋮	⋮	⋮
495	0.0565967	466	0
496	0.048756	8	0
497	0.0403025	153	0
498	0.0296797	320	0
499	0.0201765	354	0

Table 4.9.a: RSC Function of the Score Combination - System AD

Rank	s(RC(AD))	Game	W/L
0	0	255	1
1	2.5	286	1
2	3	333	1
3	3	122	1
⋮	⋮	⋮	⋮
277	277	69	0
278	277	117	1
279	277	465	1
280	277	29	1
281	279	421	0
282	280.5	197	1
⋮	⋮	⋮	⋮
495	496	8	0
496	496	466	0
497	497	153	0
498	497.5	320	0
499	497.5	354	0

Table 4.9.b: RSC Function of the Rank Combination - System AD

Rank	s(SC(AE))	Game	W/L
0	1	255	1
1	0.930107	333	1
2	0.924177	122	1
3	0.918089	162	1
⋮	⋮	⋮	⋮
277	0.563246	387	0
278	0.561332	199	1
279	0.561032	136	1
280	0.555678	465	1
281	0.555518	266	1
282	0.548666	69	0
⋮	⋮	⋮	⋮
495	0.131685	443	0
496	0.0787013	153	0
497	0.0636502	320	0
498	0.0261532	354	0
499	0.018453	466	0

Table 4.10.a: RSC Function of the Score Combination - System AE

Rank	s(RC(AE))	Game	W/L
0	0	255	1
1	3	286	1
2	3	122	1
3	3.5	333	1
⋮	⋮	⋮	⋮
277	276.5	117	1
278	279	424	1
279	280.5	136	1
280	280.5	266	1
281	281.5	465	1
282	282.5	69	0
⋮	⋮	⋮	⋮
495	492.5	228	0
496	496	153	0
497	497	320	0
498	498	466	0
499	498.5	354	0

Table 4.10.b: RSC Function of the Rank Combination - System AE

Rank	s(SC(BC))	Game	W/L
0	1	255	1
1	0.885157	122	1
2	0.852654	333	1
3	0.851758	162	1
⋮	⋮	⋮	⋮
277	0.516704	251	1
278	0.516045	281	0
279	0.516041	56	0
280	0.5153	136	1
281	0.504368	232	1
282	0.501562	266	1
⋮	⋮	⋮	⋮
495	0.0478218	466	0
496	0.0449323	320	0
497	0.0379372	354	0
498	0.0141886	8	0
499	0.00296164	153	0

Table 4.11.a: RSC Function of the Score Combination - System BC

Rank	s(RC(BC))	Game	W/L
0	0	255	1
1	1.5	122	1
2	3.5	162	1
3	4.5	333	1
⋮	⋮	⋮	⋮
277	275.5	56	0
278	276	29	1
279	277.5	281	0
280	279	136	1
281	280.5	232	1
282	281.5	266	1
⋮	⋮	⋮	⋮
495	494	466	0
496	496	320	0
497	497	354	0
498	497.5	8	0
499	498	153	0

Table 4.11.b: RSC Function of the Rank Combination - System BC

Rank	s(SC(BD))	Game	W/L
0	1	255	1
1	0.915188	122	1
2	0.883228	333	1
3	0.87014	162	1
⋮	⋮	⋮	⋮
277	0.536238	281	0
278	0.532894	232	1
279	0.532095	56	0
280	0.531149	29	1
281	0.52994	387	0
282	0.527117	266	1
⋮	⋮	⋮	⋮
495	0.0581137	354	0
496	0.0513725	320	0
497	0.0452276	466	0
498	0.0167669	8	0
499	0	153	0

Table 4.12.a: RSC Function of the Score Combination - System BD

Rank	s(RC(BD))	Game	W/L
0	0	255	1
1	1	122	1
2	2	333	1
3	3.5	162	1
⋮	⋮	⋮	⋮
277	277.5	232	1
278	277.5	281	0
279	278	56	0
280	278.5	29	1
281	279	387	0
282	281	421	0
⋮	⋮	⋮	⋮
495	495.5	354	0
496	496.5	320	0
497	496.5	466	0
498	497.5	8	0
499	499	153	0

Table 4.12.b: RSC Function of the Rank Combination - System BD

Rank	s(SC(BE))	Game	W/L
0	1	255	1
1	0.960701	122	1
2	0.943904	333	1
3	0.935326	162	1
⋮	⋮	⋮	⋮
277	0.577104	49	0
278	0.574424	26	0
279	0.565932	263	1
280	0.563562	465	1
281	0.562526	268	1
282	0.562006	232	1
⋮	⋮	⋮	⋮
495	0.113791	249	0
496	0.085343	320	0
497	0.0640904	354	0
498	0.0383988	153	0
499	0.00708389	466	0

Table 4.13.a: RSC Function of the Score Combination - System BE

Rank	s(RC(BE))	Game	W/L
0	0	255	1
1	1	122	1
2	2.5	162	1
3	2.5	333	1
⋮	⋮	⋮	⋮
277	274	49	0
278	276	26	0
279	280.5	263	1
280	280.5	232	1
281	281.5	465	1
282	282	7	0
⋮	⋮	⋮	⋮
495	494	249	0
496	496	320	0
497	496.5	354	0
498	498	153	0
499	498.5	466	0

Table 4.13.b: RSC Function of the Rank Combination - System BE

Rank	s(SC(CD))	Game	W/L
0	1	255	1
1	0.863864	122	1
2	0.837344	426	1
3	0.827023	162	1
⋮	⋮	⋮	⋮
277	0.498691	307	1
278	0.497058	199	1
279	0.495139	56	0
280	0.494132	251	1
281	0.489298	387	0
282	0.487759	62	0
⋮	⋮	⋮	⋮
495	0.0788817	466	0
496	0.0291547	320	0
497	0.0201765	354	0
498	0.00559542	8	0
499	0.00296164	153	0

Table 4.14.a: RSC Function of the Score Combination - System CD

Rank	s(RC(CD))	Game	W/L
0	0	255	1
1	1.5	122	1
2	4	162	1
3	4	286	1
⋮	⋮	⋮	⋮
277	276.5	22	1
278	278.5	199	1
279	278.5	251	1
280	278.5	56	0
281	279.5	387	0
282	281.5	62	0
⋮	⋮	⋮	⋮
495	493	254	0
496	496.5	320	0
497	497.5	354	0
498	498	153	0
499	498	8	0

Table 4.14.b: RSC Function of the Rank Combination - System CD

Rank	s(SC(CE))	Game	W/L
0	1	255	1
1	0.909377	122	1
2	0.892209	162	1
3	0.880252	333	1
⋮	⋮	⋮	⋮
277	0.54125	32	1
278	0.539496	421	0
279	0.537344	233	1
280	0.536825	499	1
281	0.532157	281	0
282	0.529212	266	1
⋮	⋮	⋮	⋮
495	0.104352	8	0
496	0.0631252	320	0
497	0.0413605	153	0
498	0.0407379	466	0
499	0.0261532	354	0

Table 4.15.a: RSC Function of the Score Combination - System CE

Rank	s(RC(CE))	Game	W/L
0	0	255	1
1	1.5	122	1
2	3	162	1
3	4.5	286	1
⋮	⋮	⋮	⋮
277	274	26	0
278	274.5	136	1
279	276	499	1
280	276.5	233	1
281	277.5	266	1
282	278	281	0
⋮	⋮	⋮	⋮
495	493	208	0
496	494.5	466	0
497	496	320	0
498	497	153	0
499	498.5	354	0

Table 4.15.b: RSC Function of the Rank Combination - System CE

Rank	s(SC(DE))	Game	W/L
0	1	255	1
1	0.939408	122	1
2	0.910826	333	1
3	0.910591	162	1
⋮	⋮	⋮	⋮
277	0.565339	29	1
278	0.561984	56	0
279	0.554767	266	1
280	0.553224	136	1
281	0.552351	281	0
282	0.550749	499	1
⋮	⋮	⋮	⋮
495	0.10693	8	0
496	0.0695654	320	0
497	0.0463297	354	0
498	0.0383988	153	0
499	0.0381437	466	0

Table 4.16.a: RSC Function of the Score Combination - System DE

Rank	s(RC(DE))	Game	W/L
0	0	255	1
1	1	122	1
2	2.5	333	1
3	3	162	1
⋮	⋮	⋮	⋮
277	273.5	465	1
278	274	56	0
279	277.5	266	1
280	278	281	0
281	278.5	136	1
282	279	499	1
⋮	⋮	⋮	⋮
495	494	249	0
496	496.5	320	0
497	497	354	0
498	497	466	0
499	498	153	0

Table 4.16.b: RSC Function of the Rank Combination - System DE

Rank	s(SC(ABC))	Game	W/L
0	1	255	1
1	0.877916	122	1
2	0.864673	333	1
3	0.856643	286	1
⋮	⋮	⋮	⋮
277	0.5225	136	1
278	0.521228	197	1
279	0.518621	387	0
280	0.513898	199	1
281	0.51202	421	0
282	0.51031	281	0
⋮	⋮	⋮	⋮
495	0.0441832	466	0
496	0.0392385	8	0
497	0.0378763	320	0
498	0.0288427	153	0
499	0.0252914	354	0

Table 4.17.a: RSC Function of the Score Combination - System ABC

Rank	s(RC(ABC))	Game	W/L
0	0	255	1
1	2.66667	122	1
2	4	286	1
3	4.33333	333	1
⋮	⋮	⋮	⋮
277	276.667	197	1
278	276.667	387	0
279	279	136	1
280	280.333	199	1
281	281.667	421	0
282	283	281	0
⋮	⋮	⋮	⋮
495	495	466	0
496	496.333	8	0
497	496.667	320	0
498	497	153	0
499	497.667	354	0

Table 4.17.b: RSC Function of the Rank Combination - System ABC

Rank	s(SC(ABD))	Game	W/L
0	1	255	1
1	0.897937	122	1
2	0.885056	333	1
3	0.873848	286	1
⋮	⋮	⋮	⋮
277	0.532688	421	0
278	0.532037	32	1
279	0.531173	197	1
280	0.530521	465	1
281	0.528793	136	1
282	0.526912	199	1
⋮	⋮	⋮	⋮
495	0.0424537	466	0
496	0.0421698	320	0
497	0.0409574	8	0
498	0.0387425	354	0
499	0.0268683	153	0

Table 4.18.a: RSC Function of the Score Combination - System ABD

Rank	s(RC(ABD))	Game	W/L
0	0	255	1
1	2.33333	122	1
2	2.66667	333	1
3	3.33333	286	1
⋮	⋮	⋮	⋮
277	278	421	0
278	278	197	1
279	278	29	1
280	279.667	465	1
281	281.333	199	1
282	281.667	136	1
⋮	⋮	⋮	⋮
495	496.333	8	0
496	496.667	354	0
497	496.667	466	0
498	497	320	0
499	497.667	153	0

Table 4.18.b: RSC Function of the Rank Combination - System ABD

Rank	s(SC(ACD))	Game	W/L
0	1	255	1
1	0.863721	122	1
2	0.852095	286	1
3	0.842622	333	1
⋮	⋮	⋮	⋮
277	0.514335	307	1
278	0.512882	281	0
279	0.512232	387	0
280	0.510926	197	1
281	0.498875	199	1
282	0.498831	69	0
⋮	⋮	⋮	⋮
495	0.0648898	466	0
496	0.0335097	8	0
497	0.0288427	153	0
498	0.0273579	320	0
499	0.013451	354	0

Table 4.20.a: RSC Function of the Score Combination - System ACD

Rank	s(RC(ACD))	Game	W/L
0	0	255	1
1	2.66667	122	1
2	3.33333	286	1
3	4.33333	333	1
⋮	⋮	⋮	⋮
277	275.667	117	1
278	276	307	1
279	276.667	281	0
280	277	197	1
281	281	69	0
282	283	199	1
⋮	⋮	⋮	⋮
495	494	466	0
496	496.667	8	0
497	497	153	0
498	497	320	0
499	498	354	0

Table 4.20.b: RSC Function of the Rank Combination - System ACD

Rank	s(SC(ACE))	Game	W/L
0	1	255	1
1	0.894063	122	1
2	0.883072	333	1
3	0.881606	162	1
⋮	⋮	⋮	⋮
277	0.541489	136	1
278	0.54127	117	1
279	0.5334	199	1
280	0.531306	499	1
281	0.528245	387	0
282	0.521052	281	0
⋮	⋮	⋮	⋮
495	0.0993474	8	0
496	0.054442	153	0
497	0.0500049	320	0
498	0.0394606	466	0
499	0.0174355	354	0

Table 4.21.a: RSC Function of the Score Combination - System ACE

Rank	s(RC(ACE))	Game	W/L
0	0	255	1
1	2.66667	122	1
2	3.66667	286	1
3	4.33333	162	1
⋮	⋮	⋮	⋮
277	275.333	117	1
278	276	136	1
279	276	199	1
280	278	499	1
281	278	387	0
282	282	266	1
⋮	⋮	⋮	⋮
495	492.667	228	0
496	495.333	466	0
497	496.333	153	0
498	496.667	320	0
499	498.667	354	0

Table 4.21.b: RSC Function of the Rank Combination - System ACE

Rank	s(SC(ADE))	Game	W/L
0	1	255	1
1	0.914083	122	1
2	0.903455	333	1
3	0.893861	162	1
⋮	⋮	⋮	⋮
277	0.550331	465	1
278	0.54895	387	0
279	0.547782	136	1
280	0.546415	199	1
281	0.540588	499	1
282	0.538067	266	1
⋮	⋮	⋮	⋮
495	0.101066	8	0
496	0.0542984	320	0
497	0.0524675	153	0
498	0.0377311	466	0
499	0.0308865	354	0

Table 4.22.a: RSC Function of the Score Combination - System ADE

Rank	s(RC(ADE))	Game	W/L
0	0	255	1
1	2.33333	122	1
2	3	333	1
3	3	286	1
⋮	⋮	⋮	⋮
277	275.667	387	0
278	277	199	1
279	277.333	465	1
280	278.667	136	1
281	280	499	1
282	282	266	1
⋮	⋮	⋮	⋮
495	492	249	0
496	497	320	0
497	497	466	0
498	497	153	0
499	497.667	354	0

Table 4.22.b: RSC Function of the Rank Combination - System ADE

Rank	s(SC(BCD))	Game	W/L
0	1	255	1
1	0.88807	122	1
2	0.85182	333	1
3	0.84964	162	1
⋮	⋮	⋮	⋮
277	0.518589	199	1
278	0.517294	136	1
279	0.517169	251	1
280	0.514425	56	0
281	0.50604	387	0
282	0.504021	232	1
⋮	⋮	⋮	⋮
495	0.0573104	466	0
496	0.0418198	320	0
497	0.0387425	354	0
498	0.0121836	8	0
499	0.00197442	153	0

Table 4.23.a: RSC Function of the Score Combination - System BCD

Rank	s(RC(BCD))	Game	W/L
0	0	255	1
1	1.33333	122	1
2	3.66667	333	1
3	3.66667	162	1
⋮	⋮	⋮	⋮
277	276.333	29	1
278	276.333	199	1
279	277.333	56	0
280	277.667	136	1
281	280.333	387	0
282	281.667	232	1
⋮	⋮	⋮	⋮
495	494.333	466	0
496	496.333	320	0
497	496.667	354	0
498	497.667	8	0
499	498.333	153	0

Table 4.23.b: RSC Function of the Rank Combination - System BCD

Rank	s(SC(BCE))	Game	W/L
0	1	255	1
1	0.918412	122	1
2	0.893097	162	1
3	0.89227	333	1
⋮	⋮	⋮	⋮
277	0.543818	421	0
278	0.541389	32	1
279	0.539524	499	1
280	0.538588	136	1
281	0.537048	197	1
282	0.536498	266	1
⋮	⋮	⋮	⋮
495	0.0780214	8	0
496	0.0644668	320	0
497	0.0427269	354	0
498	0.0318812	466	0
499	0.0275736	153	0

Table 4.24.a: RSC Function of the Score Combination - System BCE

Rank	s(RC(BCE))	Game	W/L
0	0	255	1
1	1.33333	122	1
2	3	162	1
3	4	333	1
⋮	⋮	⋮	⋮
277	272.667	56	0
278	273.333	29	1
279	277	499	1
280	277.667	266	1
281	279	197	1
282	280	136	1
⋮	⋮	⋮	⋮
495	493	249	0
496	495.667	466	0
497	496	320	0
498	497.333	354	0
499	497.667	153	0

Table 4.24.b: RSC Function of the Rank Combination - System BCE

Rank	s(SC(BDE))	Game	W/L
0	1	255	1
1	0.938432	122	1
2	0.912653	333	1
3	0.905352	162	1
⋮	⋮	⋮	⋮
277	0.55804	29	1
278	0.555587	465	1
279	0.553535	266	1
280	0.548807	499	1
281	0.546993	197	1
282	0.545694	281	0
⋮	⋮	⋮	⋮
495	0.0797403	8	0
496	0.0687603	320	0
497	0.0561779	354	0
498	0.0301517	466	0
499	0.0255992	153	0

Table 4.25.a: RSC Function of the Score Combination - System BDE

Rank	s(RC(BDE))	Game	W/L
0	0	255	1
1	1	122	1
2	2.33333	333	1
3	3	162	1
⋮	⋮	⋮	⋮
277	275	29	1
278	277.333	465	1
279	277.667	266	1
280	279	499	1
281	280.333	197	1
282	281	281	0
⋮	⋮	⋮	⋮
495	494	249	0
496	496.333	320	0
497	496.333	354	0
498	497.333	466	0
499	498.333	153	0

Table 4.25.b: RSC Function of the Rank Combination - System BDE

Rank	s(SC(CDE))	Game	W/L
0	1	255	1
1	0.904216	122	1
2	0.876608	162	1
3	0.870219	333	1
⋮	⋮	⋮	⋮
277	0.534804	281	0
278	0.534456	251	1
279	0.534351	56	0
280	0.526746	197	1
281	0.525149	421	0
282	0.524272	32	1
⋮	⋮	⋮	⋮
495	0.0722926	8	0
496	0.0539484	320	0
497	0.0525878	466	0
498	0.0308865	354	0
499	0.0275736	153	0

Table 4.26.a: RSC Function of the Score Combination - System CDE

Rank	s(RC(CDE))	Game	W/L
0	0	255	1
1	1.33333	122	1
2	3.33333	162	1
3	4	286	1
⋮	⋮	⋮	⋮
277	274.333	499	1
278	274.333	421	0
279	274.667	281	0
280	274.667	136	1
281	274.667	56	0
282	279.333	197	1
⋮	⋮	⋮	⋮
495	493.333	8	0
496	494.667	466	0
497	496.333	320	0
498	497.667	354	0
499	497.667	153	0

Table 4.26.b: RSC Function of the Rank Combination - System CDE

Rank	s(SC(ABCD))	Game	W/L
0	1	255	1
1	0.881911	122	1
2	0.861043	333	1
3	0.854865	286	1
⋮	⋮	⋮	⋮
277	0.522663	197	1
278	0.522195	136	1
279	0.519055	387	0
280	0.517757	281	0
281	0.514568	199	1
282	0.508129	421	0
⋮	⋮	⋮	⋮
495	0.0522093	466	0
496	0.037306	320	0
497	0.0314723	8	0
498	0.0290568	354	0
499	0.0216321	153	0

Table 4.27.a: RSC Function of the Score Combination - System ABCD

Rank	s(RC(ABCD))	Game	W/L
0	0	255	1
1	2.25	122	1
2	3.75	286	1
3	3.75	333	1
⋮	⋮	⋮	⋮
277	276.5	29	1
278	276.5	387	0
279	278	136	1
280	279.25	281	0
281	280.25	199	1
282	282.75	421	0
⋮	⋮	⋮	⋮
495	495	466	0
496	496.75	320	0
497	496.75	8	0
498	497.25	354	0
499	497.5	153	0

Table 4.27.b: RSC Function of the Rank Combination - System ABCD

Rank	s(SC(ABCE))	Game	W/L
0	1	255	1
1	0.904667	122	1
2	0.89138	333	1
3	0.884923	162	1
⋮	⋮	⋮	⋮
277	0.54515	421	0
278	0.540463	199	1
279	0.538166	136	1
280	0.53471	499	1
281	0.531064	387	0
282	0.529955	197	1
⋮	⋮	⋮	⋮
495	0.0808506	8	0
496	0.0542912	320	0
497	0.0408315	153	0
498	0.0331374	466	0
499	0.0320452	354	0

Table 4.28.a: RSC Function of the Score Combination - System ABCE

Rank	s(RC(ABCE))	Game	W/L
0	0	255	1
1	2.25	122	1
2	4	333	1
3	4	286	1
⋮	⋮	⋮	⋮
277	274.25	29	1
278	275	199	1
279	278.25	499	1
280	279	387	0
281	279.75	136	1
282	281	266	1
⋮	⋮	⋮	⋮
495	493.25	8	0
496	496	466	0
497	496.5	320	0
498	497	153	0
499	497.75	354	0

Table 4.28.b: RSC Function of the Rank Combination - System ABCE

Rank	s(SC(ACDE))	Game	W/L
0	1	255	1
1	0.894021	122	1
2	0.874842	333	1
3	0.872556	162	1
⋮	⋮	⋮	⋮
277	0.534235	32	1
278	0.532574	499	1
279	0.531148	421	0
280	0.529195	199	1
281	0.526272	387	0
282	0.525813	281	0
⋮	⋮	⋮	⋮
495	0.076554	8	0
496	0.0486673	466	0
497	0.0464024	320	0
498	0.0408315	153	0
499	0.0231649	354	0

Table 4.30.a: RSC Function of the Score Combination - System ACDE

Rank	s(RC(ACDE))	Game	W/L
0	0	255	1
1	2.25	122	1
2	3.5	286	1
3	4	333	1
⋮	⋮	⋮	⋮
277	274	117	1
278	275.75	136	1
279	276.25	499	1
280	277	199	1
281	277.5	387	0
282	279.5	281	0
⋮	⋮	⋮	⋮
495	493.5	8	0
496	495.25	466	0
497	496.75	320	0
498	497	153	0
499	498	354	0

Table 4.30.b: RSC Function of the Rank Combination - System ACDE

Rank	s(SC(BCDE))	Game	W/L
0	1	255	1
1	0.912283	122	1
2	0.88174	333	1
3	0.881174	162	1
⋮	⋮	⋮	⋮
277	0.538738	499	1
278	0.534528	197	1
279	0.534262	136	1
280	0.534198	281	0
281	0.531978	421	0
282	0.528621	32	1
⋮	⋮	⋮	⋮
495	0.0605595	8	0
496	0.0572488	320	0
497	0.0429828	466	0
498	0.0421335	354	0
499	0.0206802	153	0

Table 4.31.a: RSC Function of the Score Combination - System BCDE

Rank	s(RC(BCDE))	Game	W/L
0	0	255	1
1	1.25	122	1
2	3.25	162	1
3	3.5	333	1
⋮	⋮	⋮	⋮
277	274.75	56	0
278	274.75	421	0
279	275.5	499	1
280	277.75	197	1
281	277.75	281	0
282	278.75	136	1
⋮	⋮	⋮	⋮
495	494.25	8	0
496	495.5	466	0
497	496.25	320	0
498	497	354	0
499	498	153	0

Table 4.31.b: RSC Function of the Rank Combination - System BCDE

Rank	s(SC(ABCDE))	Game	W/L
0	1	255	1
1	0.902513	122	1
2	0.883135	333	1
3	0.87702	162	1
⋮	⋮	⋮	⋮
277	0.535686	199	1
278	0.535411	421	0
279	0.535043	499	1
280	0.534789	136	1
281	0.529358	197	1
282	0.528923	387	0
⋮	⋮	⋮	⋮
495	0.0663152	8	0
496	0.050552	320	0
497	0.0417674	466	0
498	0.0337068	354	0
499	0.0326652	153	0

Table 4.32.a: RSC Function of the Score Combination - System ABCDE

Rank	s(RC(ABCDE))	Game	W/L
0	0.166667	255	1
1	1.83333	122	1
2	3.16667	333	1
3	3.33333	286	1
⋮	⋮	⋮	⋮
277	229.167	29	1
278	230.167	199	1
279	230.833	499	1
280	232	387	0
281	232.5	136	1
282	233.167	197	1
⋮	⋮	⋮	⋮
495	411.833	8	0
496	413.167	466	0
497	413.833	320	0
498	414.5	354	0
499	414.5	153	0

Table 4.32.b: RSC Function of the Rank Combination - System ABCDE

Abstract

Henry William Gorelick

BA, University of Michigan

MS, Fordham University

Predicting and Enhancing Hearthstone Strategy with Combinatorial Fusion

Master's Thesis directed by D. Frank Hsu, Ph.D.

The goal of this master's thesis is to demonstrate that combinatorial fusion analysis (CFA) can effectively predict winners and enhance play strategy of Blizzard Entertainment's collectible card game Hearthstone. CFA is used to combine and evaluate the performance of the combinatorial combinations of five machine learning models trained on 500 Hearthstone game simulations. For each combinatorial combination, the score function of the score combination and the score function of the rank combination is derived for each of the five models, and the performance of each is compared and evaluated. The improvement in performance of certain combinations over the individual components validates that CFA is an effective method for predicting the winner of Hearthstone games and enhancing play strategy. Furthermore, the resulting models could be used to boost Monte Carlo Tree Search and implement a competitive Hearthstone playing AI agent.

Keywords – **Combinatorial Fusion Analysis, Rank-Score Characteristic (RSC) Function, Cognitive Diversity, Hearthstone, machine learning, AI, Monte Carlo Tree Search**

Vita

Henry William Gorelick, son of Todd and Stacy Gorelick, was born on April 26, 1995 in Charlotte, North Carolina. After graduating from Charlotte Country Day School in 2013, Henry attended the University of Michigan in Ann Arbor. Henry graduated from U of M in 2017 with a Bachelor of Arts degree in English Language & Literature.

From July 2017 to September 2019, he worked as a technical writing intern, and then robotics and automation researcher for the Long Island City based manufacturing firm Boyce Technologies, Inc. During his time at Boyce, Henry was awarded a graduate assistantship at Fordham University where, in August of 2018, he began working towards a Master of Science degree in computer science. Under the guidance of Dr. D. Frank Hsu, Henry completed a master's thesis prior to graduating in May 2020.