Grouping Method for Faster Hodgerank Algorithms

A thesis submitted to
Loyola Marymount University
The Mathematics Department
in partial fulfillment of the requirements
for Graduation with the Bachelor of Science Degree

by

Shelby Ferrier

May 2022

Grouping Method for Faster Hodgerank Algorithms

Senior Thesis by

Shelby Ferrier

Junyuan Lin, Thesis Director

Abstract:

In this research, we are interested in developing fast algorithms to find the statistical ranking on data that is highly incomplete and unbalanced. Based on the HodgeRank algorithm, we can describe the ranking problem on graphs and define the least square problem that measures the reliability of the ranking. We apply the HodgeRank algorithm as well as the naive ranking method on various data sets including COVID-19 symptomatic data and NBA game scores to examine their efficiency and accuracy. As the size of the data set becomes larger, it gets more expensive to compute the ranking accurately. This motivates us to design approximation algorithms that reduce the graph size and efficiently compute the ranking. By grouping elements based on their tier in a naive ranking, we can run the HodgeRank algorithm on smaller groups, which makes the method faster, while maintaining the intergrity of ranking. We examine the efficacy and the time complexity of the proposed algorithm with different selections on the number of groups.

Thesis written by

Shelby Ferrier

Approved by

Junyuan Lin, Thesis Director Date

Lily Khadjavi, Mathematics Department Chair Date

Contents

Chapter 1. Introduction 1

Chapter 2. Method and Model 2
2.1. Terminology 2
2.2. Graph Building 3
2.3. Least Squares Problem 4
2.4. Assessing Inconsistencies in the Graph 5
2.5. Hodge Decomposition 5
2.6. Solving with Linear Algebra 6
2.7. Methods to reduce run time 7

Chapter 3. Experimentation 8
3.1. Ranking Symptoms of COVID-19 8
3.2. Analysis on 2021 National Basketball Association (NBA) game data 11

Chapter 4. Grouping Method 14
4.1. Grouping Method 14
4.2. 2021 NBA season 17
4.3. Time complexity 19

Chapter 5. Conclusion 20

Chapter 6. Acknowledgements 21

Bibliography 22

CHAPTER 1

Introduction

Statistical Ranking problems are central to a wide range of applications, including sports, web search, literature search. Over the years, researchers have been searching for robust algorithms that can obtain accurate ranking [4, 6, 11, 14, 17, 18, 2, 5, 12]. Out of those methods, the HodgeRank algorithm, proposed by Jiang et al. [16], derives a cardinal ranking from subjective, incomplete data sets which contain voters (e.g. people who have reviewed some movies) and scored elements (e.g. the ratings each person gives to a movie). The HodgeRank algorithm is distinguished from other ranking methods by its ability to take into account the bias that each voter may have. For example, some people give most movies five stars, whereas others may have higher standards for what a good movie is. The algorithm resolves this by focusing on the difference one voter gives between two different elements, as opposed to their individual ratings. As an aside, only voters who have rated two or more elements in the set will have their data affect the final scoring.

HodgeRank is particularly useful on data sets where bias of the voters may need to be considered, as well as data sets that are incomplete; that is, not every voter rates every element. We include a detailed summary of Jiang et al.'s formation of HodgeRank in Chapter 2. In Chapter 3 we apply the HodgeRank algorithm to two datasets: ranking COVID-19 symptoms and NBA teams.

HodgeRank involves taking the pseudo inverse of an $n \times n$ matrix where n is the number of elements to be ranked. As the set of elements grows, the run time of the algorithm increases with time complexity of $O(n^3)$. We summarize some algorithms including the Algebraic Multigrid (AMG) method [3, 9] that cut down on run time to $O(n \log n)$ while closely approximating the universal ranking.

In Chapter 4 we propose a new method to reduce the time complexity of HodgeRank by calculating the ranking on smaller groups, as opposed to the entire set. We tested the grouping-based method on NBA team data and found high correlation between the results and the ranking found using the original HodgeRank algorithm. Finally, we analyze the time complexity of the method.

CHAPTER 2

Method and Model

The goal of the HodgeRank algorithm is to obtain a relative ranking of elements in a set, based off of ratings given to them by individual voters.

To demonstrate the method created by Jiang et al. [16], we take an example through the method. Suppose we want to rate the symptoms of COVID-19 by severity, thus we survey three patients on a set of symptoms (fever, sore throat, cough, and nausea):

	Fever	Sore Throat	Cough	Nausea
Patient 1	3	2	2	5
Patient 2	7	8	9	X*
Patient 3	2	2	1	3

*The 'X' in the above data set represents a missing value.

Hypothetically, one could take the average of each column and get each symptom's average score, weighted by the number of people who reviewed it. Upon further inspection, doing this results in every symptom scoring a four. These results are not very satisfying as one can reasonably predict that certain symptoms should be rated higher than others: nausea, for one, is rated the most severe by both people who reviewed it. To get a more accurate ranking, we implement HodgeRank instead.

2.1. Terminology

There is some terminology needed to understand the method, which we detail here. Following the notation used in Jiang et al. [16] and Colley et al. [3], we define Λ to be the set of voters and V to be the set of elements that are voted on. For $\alpha \in \Lambda$, we denote V_α to be the set of elements rated by voter α. Similarly, we let Λ_{ij} denote the set of voters who rated both elements i and j.

In our example, the following is the case:

$$\Lambda = \{\text{Patient 1, Patient 2, Patient 3}\}$$

$$V = \{\text{Fever, Sore Throat, Cough, Nausea}\}$$

$$V_{\text{Patient 2}} = \{\text{Fever, Sore Throat, Cough}\}$$

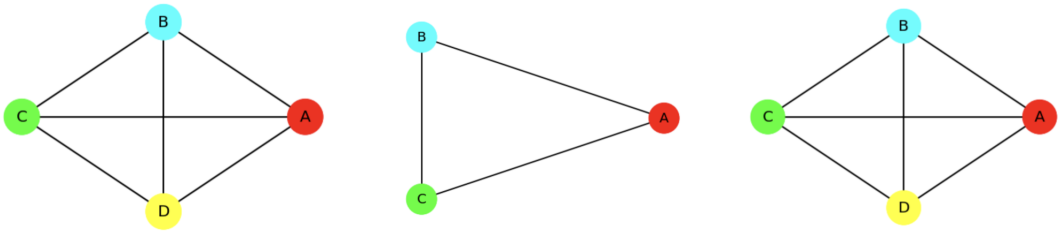
Also, we define the rankings as $R : \Lambda \times V \to \mathbb{R}$. For example, if voter α gave element i a score of 5, we would say $R(\alpha, i) = 5$.

Using this terminology, we find a universal rating.

2.2. Graph Building

This method involves building a complete graph with $|V|$ nodes that encodes information from our entire data set. In graph theory, a complete graph is a graph which has an edge connecting every pair of nodes. To build the graph that pertains to the entire data set, we create graphs for every voter.

Using the elements in V_α as nodes, we form a complete graph for every voter where each node represents an element. Note in the example below we replace the name of each symptom (fever, sore throat, cough, nausea) with letters (A, B, C, D), respectively.

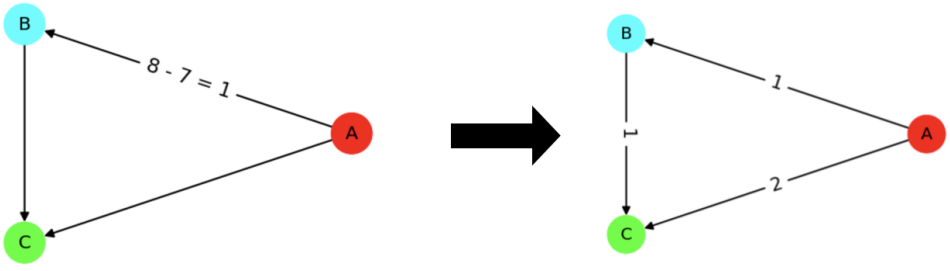


Patient 1 Patient 2 Patient 3

Let E denote the set of all edges. For every edge, we define an orientation by indiscriminately designating one node to be the sink node and the other to be the source node. To keep things simple, we let nodes which are indexed earlier be the source nodes.

The relationship between pairs of nodes is described with the pairwise comparison function, $f^\alpha(i, j)$ where α is a voter.

$$f^\alpha(i, j) = R(\alpha, j) - R(\alpha, i) \tag{1}$$



Patient 2's graph with pairwise comparisons

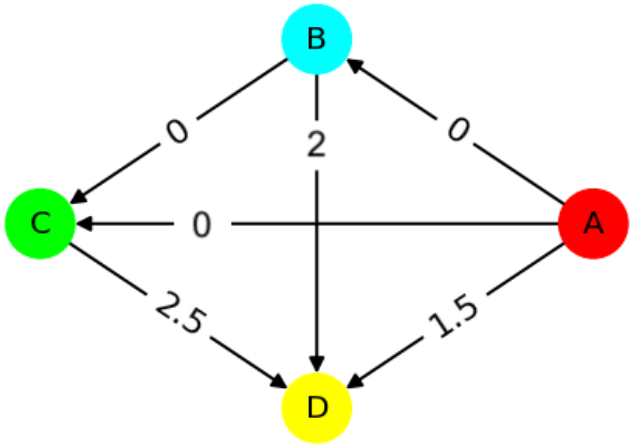
We can now define one graph, G, pertaining to all voter's data. G is a complete graph containing every alternative in V, so long as it's been voted on, as well as $\binom{|V|}{2}$ edges.

We should take into higher consideration pairs of elements which were voted on by many people. With this in mind, we define the weights for each edge as the number of voters who rated both alternatives:

$$\omega_{ij} = |\Lambda_{ij}| \tag{2}$$

where Λ_{ij} is the set of voters who rated both i and j. The edge flow of the entire group's graph, $f : V \times V \to \mathbb{R}$, represents the average pairwise difference of each edge:

$$f(i,j) = \frac{1}{|\Lambda_{ij}|} \sum_{\alpha \in \Lambda_{ij}} f^{\alpha}(i,j) \tag{3}$$



Later in this paper we'll refer to the edge flow as the vectorized version of f, indexed by the set of edges E, such that $\vec{f} \in \mathbb{R}^{|E|}$.

2.3. Least Squares Problem

Our goal is to find a universal rating $r : V \to \mathbb{R}$ that maps every element to its relative rating. By comparing r to the data processed in our graph, we can judge the efficacy of our rating. A good choice for r should agree highly with our edge flow, so we must minimize:

$$f(i,j) - (r(j) - r(i)) \tag{4}$$

While minimizing Equation 4, we should also take into account the number of voters who evaluated both i and j. If a lot of voters evaluated both elements, we should take that edge into greater consideration than if not many evaluated it. This handles the imbalanced nature of our data.

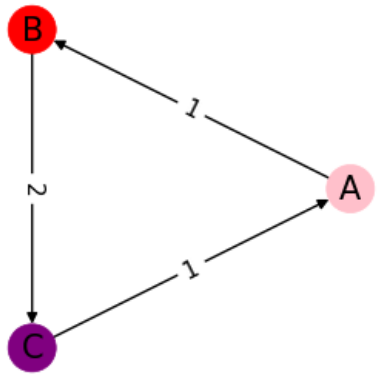
With this in mind we arrive at the function we use to judge the efficacy of any r:

$$\sum_{i,j \in V} \omega_{ij}(f(i,j) - (r(j) - r(i))^2 \tag{5}$$

Notably, multiplying by ω_{ij} does not boost any element's rating. Instead it places more importance (or lack thereof) on the balance of $f(i,j)$ and the difference of the universal ratings.

2.4. Assessing Inconsistencies in the Graph

In many data sets, there will be contradictions in the graph; it is not uncommon to find paradoxical cycles in G, especially when each voter only reviews some of the elements in V. Let the figure below be a graph processed using Hodge, where the values along the edges are edgeflows.



In the above graph, the edge flows suggest that element A should be scored lower than element B, which should be scored lower than element C, which should be scored lower than element A. Clearly, defining a perfect universal ranking in this scenario is impossible. In their paper proposing the HodgeRank algorithm, Jiang et al. [16] detail a way to quantify the extent of the local inconsistencies in a vector which we call $\vec{c}$. A detailed derivation of $\vec{c}$ is not shown in this paper, but for interested readers we provide the basic algorithm in 2.6.

To lay the groundwork for future sections, $\vec{c}$ is be indexed by T where T is the set of all 3-cycles in the graph G. Similarly to the edges, we arbitrarily define an orientation for each 3-cycle in T.

2.5. Hodge Decomposition

In this section we demonstrate how Hodge decomposition is used to show that it is possible to find a ranking $\vec{r} \in \mathbb{R}^{|V|}$ (where $\vec{r}$ is the vectorized form of r indexed by V) and local consistency $\vec{c} \in \mathbb{R}^{|T|}$ for any $\vec{f} \in \mathbb{R}^{|E|}$.

First we must define the boundary operators of the graph:

- The negative divergence, denoted by $\partial_1 : \mathbb{R}^{|E|} \to \mathbb{R}^{|V|}$, is defined as:

$$(\partial_1)_{ij} = \begin{cases} -1, & \text{if } v_i \text{ is the source node in } e_j, \\ 1, & \text{if } v_i \text{ is the sink node in } e_j, \\ 0, & \text{else.} \end{cases}$$

  Note the divergence is simply ∂_1^T.

- The curl, $\partial_2 : \mathbb{R}^{|T|} \to \mathbb{R}^{|E|}$, is defined as:

$$(\partial_2)_{ij} = \begin{cases} 1, & \text{if } e_i \in T_j \text{with same orientation as } T_j, \\ -1, & \text{if } e_i \in T_j \text{with same orientation as } T_j, \\ 0, & \text{else.} \end{cases}$$

The 1-Hodge Laplacian is $L_1 = \partial_1^T \partial_1 + \partial_2 \partial_2^T$. Due to Hodge decomposition [3, 16, 19], we can show the following:

$$\mathbb{R}^{|E|} = im(\partial_1^T) \oplus ker(L_1) \oplus im(\partial_2^T)$$

Therefore for any $\vec{f} \in \mathbb{R}^{|E|}$ we can find $\vec{r} \in \mathbb{R}^{|V|}$, $\vec{c} \in \mathbb{R}^{|T|}$, and $\vec{x_h} \in ker(L_1)$ such that:

$$\vec{f} = \partial_1^T \vec{r} + \partial_2 \vec{c} + \vec{x}_h$$

This shows that for any $\vec{f}$ we'd be able to find a ranking $\vec{r}$ and a local consistency $\vec{c}$. An extensive explanation of Hodge decomposition can be found in Lim et al.'s work [19], with implementations demonstrated in [3, 16].

2.6. Solving with Linear Algebra

Using linear algebra the minimization problem in 2.3 can be rewritten. First, let $\vec{\omega}$ be the vectorized version of ω indexed by E. Using the negative divergence of the system, the minimization becomes:

$$\min_{\vec{r} \in \mathbb{R}^{|\Lambda|}} ||\vec{f} - \partial_1^T \vec{r}||_W^2 \tag{6}$$

where W is the diagonal matrix whose entries are $\vec{\omega}$.

Using some basic calculus, the minimization reduces to the following:

$$\partial_1 W \partial_1^T \vec{r} = \partial_1 W \vec{f} \tag{7}$$

The only unknown in this equation is $\vec{r}$ so this is an $A\vec{x} = \vec{b}$ problem. The matrix $\partial_1 W \partial_1^T$ is a well studied matrix called the graph Laplacian, which has no inverse, so when solving for $\vec{r}$ the pseudo-inverse must be taken. The time complexity of this operation is $O(n^3)$ where $n = |V|$.

Without detailing the derivation, the solution of $\vec{c}$ follows the same pattern. The minimization problem reduces to:

$$\min_{\vec{c}\in\mathbb{R}^{|E|}} ||\vec{f} - \partial_2\vec{c}||_W^2 \tag{8}$$

Similarly, this reduces to:

$$\partial_2^T W \partial_2 \vec{c} = \partial_2^T W \vec{f}, \tag{9}$$

This equation has one unknown, $\vec{c}$, which can be solved for with complexity $O(n^3)$, where $n = |E|$.

2.7. Methods to reduce run time

The usability of this method is impacted by the potentially great computational run time. The most computationally expensive step of the method is taking the pseudo-inverse of the graph Laplacian which is an operation of order $O(n^3)$ where n is $|V|$.

There are a few established methods to reduce the cost of solving for $\vec{r}$ in $\partial_1 W \partial_1^T \vec{r} = \partial_1 W \vec{f}$.

One is the Algebraic Multigrid (AMG) Method, which lets $\vec{x} \in \mathbb{R}^n$ be approximated with linear complexity, $O(n)$, where $\vec{x}$ is the only unknown in $A\vec{x} = \vec{b}$. Furthermore, the work can be done in parallel across multiple machines, making it an ideal choice for implementing HodgeRank when the number of elements to be ranked is very large. In short, the method is a successive subspace correction method which recursively partitions the solution space to approximate the best solution. More information can be found in the article by Falgout et al. [9] which introduces the method, as well as in the paper by Colley et al. [3] which implements AMG directly with HodgeRank.

Tai et al. [22] detail a successive subspace correction method (SSC), which is a general convex optimization algorithm that decomposes the original problem into a number of smaller optimization problems.

Additionally there is a method created by Drineas et al. [7] [8] which employs a row sampling method to approximate large-scale matrix multiplication and reduce graph size.

We introduce a new method in Chapter 4 that falls under the umbrella of dimensional reduction and is specifically suited to reduce the run time of the least squares solver on universal ranking problems.

CHAPTER 3

Experimentation

We implemented HodgeRank in Python using Pandas, which is a Python package used for data manipulation [21, 23] and NumPy [15] which we used for most of our mathematical operations. We experimented with the HodgeRank on two data sets: COVID-19 symptom data and National Basketball Association (NBA) game data. We detail the pre-processing required and final results for each data set below.

3.1. Ranking Symptoms of COVID-19

The COVID-19 symptoms tracker released by The Centre for Evidence-Based Medicine at Oxford University [20] contains data collected by 5700 people who have had COVID, of whom 1376 had a severe case and 4324 had a non-severe case. The researchers defined a severe case of COVID as meeting one or more of the following criteria: "1) presence of shortness of breath with a respiratory rate $\geq$ 30 breaths/minute; 2) an oxygen saturation (SpO2) 93% in the resting state; 3) hypoxemia defined as an arterial partial pressure of oxygen divided by the fraction of inspired oxygen (PaO2/FiO2 ratio) $\leq$ 300 mmHg; [or] 4) presence of radiographic progression, defined as $\geq$ 50% increase of target lesion within 24-48 hours." [20]

With this data, the set of voters, Λ, is the set of patients, and the group of elements being ranked, V, are the symptoms.

The data set gives the percentage of people with severe and non-severe COVID who experienced each symptom:

	Severe	Non-severe
Fever	88.4%	81.4%
Cough	71.1%	65.7%
Fatigue	60.3%	44.2%
Dyspnea	44.2%	5.7%
Sputum production	37.6%	28%
Shortness of breath	35.7%	12.8%
Myalgia	26%	13.1%
Chill	26%	10.9%
Dizziness	16.1%	12.1%
Headache	11.3 %	13.5%
Sore throat	7.8%	9.7%
Nausea or Vomiting	5.9 %	5.7%
Diarrhea	5.7%	5.8%
Nasal congestion	2.8%	5.1%

This data set contains data for large groups of people, as opposed to individual voters, as the method requires. To resolve this, we generated data for each of the 5700 patients.

To generate individual patient data, we started by initializing two data frames for severe and non-severe patients. The columns of the data frames represent symptoms and the rows represent individual patients. Thus, the dimensions of the severe data frame was 1376 rows by 14 columns and each row gave the hypothetical ratings one patient had of all 14 COVID symptoms. In each data frame we reflected the percent values in our original data set by randomly filling the columns of each symptom with the appropriate percentage of non-zero values, setting all other values equal to zero. After doing this, each data frame contained cells of only two values: non-zero constants and zeros.

In the real world, every patient has different rates of severity of each of their symptoms, therefore it's inaccurate to give each patient the same rating of every symptom they had. Therefore, we randomly distributed each patient's symptoms around a normal distribution. We decided to let the patient's non-zero scores fall between 1 and 10, similar to the 0-10 scale used for rating pain levels.

To accomplish this, for patients with severe symptoms, we centered their distribution around 8 with a standard deviation of 2. With this distribution, most severe patients will have a relatively high average symptoms rating, with some of their symptoms being quite bad (~ 10), and many others being more manageable.

For patients with non-severe symptoms, we wanted their average score to be quite low, so we centered their non-zero symptoms around 3. Patients with non-severe symptoms likely wouldn't have many symptoms in the 5-10 range. Therefore, non-severe patient's non-zero values had a standard deviation of 1, so that about 95% of values would be less than 5.

The complete data set used in our calculations can be found in our GitHub repository for this project [10].

We ran this data through the HodgeRank algorithm which we wrote in Python (also available on [10]) and got the below universal rating of symptoms. Note that we added an arbitrarily chosen constant, 0.656, to each value of $\vec{r}$ to make all ratings positive. Doing this does not affect the final error as the constants cancel out.

Symptom	HodgeRank rating ($\vec{r}$)
Fever	2.186
Cough	1.936
Fatigue	1.531
Sputum Production	1.156
Dyspnea	0.906
Chills	0.875
Shortness of Breath	0.812
Myalgia	0.781
Dizziness	0.6873
Headache	0.656
Nausea	0.625
Sore Throat	0.562
Diarrhea	0.5
Congestion	0.5

For context, we compare this ranking to one that might be created from a more naive ranking method. For the naive score, we let each symptom's rating be the percent of people who presented the symptoms:

Symptom	HodgeRank rating ($\vec{r}$)	Naive rating
Fever	2.186	0.831
Cough	1.936	0.6700
Fatigue	1.531	0.4809
Sputum Production	1.156	0.3032
Dyspnea	0.906	0.1499
Chills	0.875	0.1454
Shortness of Breath	0.812	0.1832
Myalgia	0.781	0.1621
Dizziness	0.6873	0.1307
Headache	0.656	0.1296
Nausea	0.625	0.5737
Sore Throat	0.562	0.9228
Diarrhea	0.5	0.5772
Congestion	0.5	0.4561

Notably, shortness of breath are both higher in the HodgeRank (HR) rating than in the simple rating, whereas dyspnea (difficulty breathing) and chills are lower. Sore throat, diarrhea, and nausea also shifted somewhat, but because their scores are relatively close to each other in both ratings, this is not of the most interest.

The main difference between these two rankings is that HodgeRank takes into account the severity of the symptoms, giving more weight to symptoms typically associated with severe COVID than non-severe. When discussing the worst symptoms of COVID-19 it is important to consider the severity of the infection so the HR rating arguably reflects the real world better than the simple rating.

There are limitations to our implementation of this data set that may affect the rigor of our final results. Randomly distributing non-zero values across columns may not closely approximate what real patient data looks like. Randomly distributing non-zero values causes the number of symptoms per patient to approach a normal distribution but it is possible that real patient data does not have this quality. Perhaps there were large groups of patients who presented many symptoms more than the average and another large group who presented few symptoms. Because HodgeRank uses the pairwise difference, this may skew the results.

Furthermore, putting each person's non-zero ratings on a normal distribution would not predict which symptoms might be rated worse than other symptoms by individuals experiencing both. Therefore, symptoms that more people tend to get may be ranked higher than slightly less common symptoms, even if the less common symptoms are experienced to be worse.

Although there were limitations with the data set used for the experiment, the rating derived from the HodgeRank algorithm is likely more accurate than the simple rating. The HodgeRank rating takes into account the severity of the patient's illness while minimizing the error of the rating. Using Equation 5 as the error metric, we can calculate the performance of each rating. The error of the HodgeRank rating is $\approx 3.2 * 10^{-24}$ and the error for the naive rating is ≈ 640000. Note that we linearly adjusted the simple rating to minimize the error. Even with this adjustment, the HodgeRank rating outperforms the naive rating.

3.2. Analysis on 2021 National Basketball Association (NBA) game data

In this section, we use the method to analyze game data from the 2020 - 2021 NBA season which detailed the results of 1080 games between 30 teams. We used data posted in a Kaggle dataset that was originally gathered from the NBA statistics website [1]. The data points we used for every game are the teams playing and the total points scored by both teams.

In order to get a universal rating of the teams, one might let each team's score be the percent of wins out of total games they played. However, a team's number of wins highly depends on the opposing teams it played in a season. Playing against the toughest teams could lower a teams rating whereas playing against the lowest ranked teams could boost it. Thus, a more sophisticated rating system is necessary to accurately reflect the performance of the teams.

To use HodgeRank we need a set of voters and a set of elements to be ranked. Immediately, we know to let the set of elements to be ranked be the set of teams. A team can said to be "rated" when it plays against another team: the number of points the home team scores above or below the visiting team can be said to be a rating of the home team, made by the visiting team. In this way every game gives us

two pieces of data that we can give to HodgeRank: the visiting team's rating of the home team and vice versa.

To fit this methodology into the mathematical framework of HodgeRank, let $G =$\{set of games\}, and $G_{ij} =$ \{set of games played between team i and j\}. Furthermore let g represent a particular game between i and j such that g_i denotes team i's score in that game. With these definitions in place, we can define our rating function as:

$$R(i,j) = \sum_{g \in G_{i,j}} g_j - g_i$$

Below we show the results we get from running this algorithm using the above definitions. For comparison, we also calculated a naive ranking, where the score of each team is the average of its point difference across all games:

$$r_n(i) = \frac{\sum_{t \in \Lambda} R(t,i)}{\sum_{t \in \Lambda} |G_{t,i}|}$$

HodgeRank

Rank	Team	r
1.	Suns	7.556
2.	Warriors	6.4924
3.	Jazz	5.6096
4.	Grizzlies	4.8294
5.	Celtics	4.4366
6.	Heat	4.3354
7.	Mavericks	4.2256
8.	Bucks	3.1643
9.	Timberwolves	2.894
10.	Cavaliers	2.3644
11.	Bulls	2.3517
12.	Nuggets	2.1021
13.	76ers	1.7672
14.	Raptors	1.7024
15.	Hawks	1.2825
16.	Nets	0.101
17.	Spurs	0.0458
18.	Knicks	-0.3068
19.	Clippers	-1.2285
20.	Hornets	-1.2994
21.	Pelicans	-2.1836
22.	Pacers	-2.2519
23.	Lakers	-3.1923
24.	Wizards	-3.2635
25.	Kings	-4.2364
26.	Trail Blazers	-6.1999
27.	Magic	-6.8426
28.	Thunder	-7.307
29.	Pistons	-8.1996
30.	Rockets	-8.7492

Naive Rank

Rank	Team	r_n
1.	Suns	8.2817
2.	Warriors	6.7945
3.	Jazz	6.0141
4.	Grizzlies	5.2703
5.	Celtics	5.125
6.	Heat	4.5467
7.	Mavericks	4.2535
8.	Bucks	3.4658
9.	Timberwolves	3.1781
10.	Bulls	2.7183
11.	Cavaliers	2.4028
12.	Nuggets	2.3151
13.	76ers	1.8841
14.	Raptors	1.625
15.	Hawks	0.7857
16.	Spurs	0.137
17.	Nets	-0.0563
18.	Knicks	-0.3099
19.	Clippers	-1.3836
20.	Hornets	-1.4722
21.	Pacers	-2.5417
22.	Pelicans	-2.6901
23.	Lakers	-3.1528
24.	Wizards	-3.1571
25.	Kings	-4.3425
26.	Trail Blazers	-7.2571
27.	Magic	-7.3889
28.	Thunder	-7.8571
29.	Pistons	-8.831
30.	Rockets	-9.2394

Using the metric described above in equation (4), we calculate the error of the HodgeRank ranking to be $\approx$ 37727.69 while the error of the naive ranking is $\approx$ 37837.91 which is greater than that of HodgeRank. This is the result we expected, as the algorithm works to minimize this value. Notably, both errors are quite a lot larger than the errors found in Section 3.1. It's likely this is due to a combination of factors. For one, the NBA team data set contains more elements to be ranked than the COVID-19 data set which increases the dimensions of $\vec{f}, \vec{r}$, and W. Additionally, the difference between team scores is often much greater than the difference between any one person's symptoms ratings. Finally, randomly generating patient data may have resulted in a graph that would be highly agreeable to a universal ranking, resulting in a very small minimum possible error.

Interestingly, neither ranking reflects the final bracket positions of the thirty teams. In the 2020 - 2021 NBA season, the Bucks earned the top place in the bracket, beating the Suns, but our ranking places the Bucks at eighth. This demonstrates the difference between HodgeRank and other scoring systems; HodgeRank says that overall, the Suns played the best even though the Bucks won the championship.

In the next chapter we experiment with this data set further to see how we can improve the run time of HodgeRank while maintaining accuracy.

CHAPTER 4

Grouping Method

In this chapter we examine a method that reduces the computational cost of the method by reducing the size of the matrix that we take the pseudo inverse of.

The key to this method is that running the entire data set through the algorithm in smaller groups will reduce our run time. An in depth description of the method is given in the next section.

4.1. Grouping Method

The first step in the grouping method is to obtain a naive ranking of elements, which we denote $r_0 : V \to \mathbb{R}$. There are various choices you could make for this first step.

1. **Arithmetic mean of rating:** In this ranking elements are ordered by their average rating. This is similar to sorting search results by "top rated".
$$r_0(i) = \frac{\sum_{\alpha \in \Lambda_i} R(\alpha, i)}{|\Lambda_i|} \tag{10}$$
2. **Arithmetic mean of edge flow:** Here r_0 is the result of averaging the edge flow between one node and every other node. The edge flow refers to f which we defined in Equation 3.
$$r_0(i) = \frac{1}{|V|} \sum_{j \in V} f(j, i) \tag{11}$$
3. **Weighted arithmetic mean of edge flow:** Here r_0 is the average edge flow between one node and every other node weighted by ω.
$$r_0(i) = \frac{\sum_{j \in V} \omega_{ij} f(j, i)}{\sum_{j \in V} \omega_{ij}} \tag{12}$$
Defining the naive rank as the weighted arithmetic mean of the edge flow yielded better empirical results, as we'll discuss later. We assume this is the case because it incorporates the edge flow and the edge weight which are both key components of the final step of HodgeRank.

Next, we evenly split the group into k subgroups by their naive rank; that is, the highest scoring elements and lowest scoring elements are kept together. It's

$$V = [v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9] \Longrightarrow \begin{array}{l} V_1 = \{v_1, v_2, v_3\} \\ V_2 = \{v_4, v_5, v_6\} \\ V_3 = \{v_7, v_8, v_9\} \end{array}$$

FIGURE 1. Example: The elements in V which are indexed by their naive ranking are split into 3 subgroups

possible to run HodgeRank on each of the groupings to achieve a universal rating, but doing so omits much data.

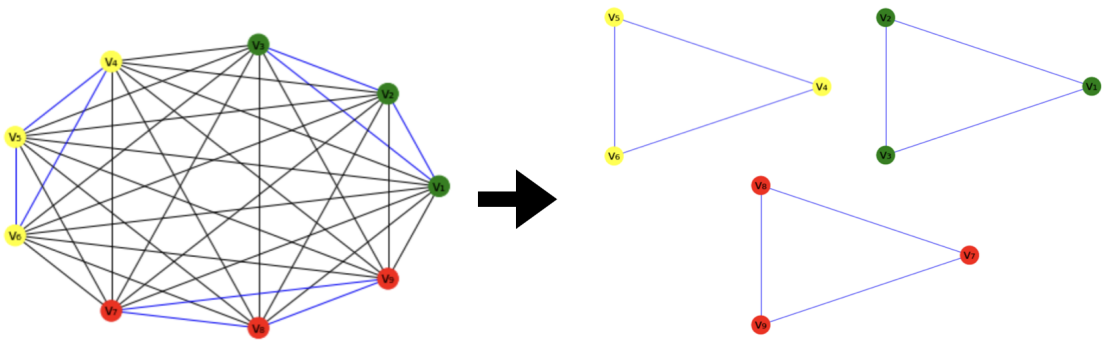


FIGURE 2. Running HodgeRank on smaller groups; in gray are all edges that are omitted

To demonstrate this, we'll count the number of edges omitted. The graph that might be formed in the normal HodgeRank algorithm has n nodes and $\binom{n}{2}$ edges. Splitting the elements into k groupings and building graphs for each of the groupings with $\lceil \frac{n}{k} \rceil$ nodes, results in a total of at most $k\binom{\lceil \frac{n}{k} \rceil}{2}$ edges. The number of edges that would be dropped is:

$$\binom{n}{k} - k\binom{\lceil \frac{n}{k} \rceil}{2} \approx \frac{n(n-1)}{2} - \frac{n(\frac{n}{k}-1)}{2} = \frac{n(n-\frac{n}{k})}{2}$$

Ideally, every edge should have an effect on our final ranking. To resolve this issue we introduce pseudo-nodes into each subgroup, which will be placeholders for subgroups connections to other subgroups.

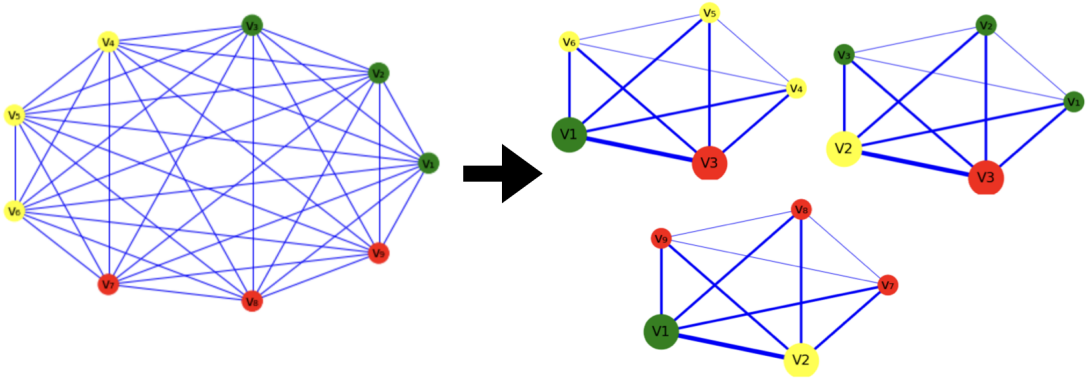


FIGURE 3. Groupings with pseudo-nodes; in this figure $V1$ represents the elements in the first group $\{v_1, v_2, v_3\}$, and so on. Edges on the right represent all edges between respective nodes, with thicker edges representing more edges from the original graph.

Let V be the set of elements and $\{V_1, V_2, \ldots, V_k\}$ be the set of subgroups. Then we let W_n for $n \in 1, 2, \ldots, k$ denote the set of nodes that HodgeRank will run on such that $W_n = \{v, V_m | v \in V_n, m \neq n\}$. We modify the definitions of edge flow and edge weight to suit the introduction of subgroups as pseudo-nodes:

$$f(i,j) = \begin{cases} \frac{1}{|\Lambda_{ij}|} \sum_{\alpha \in \Lambda_{ij}} f^{\alpha}(i,j) & \text{if } i \text{ and } j \text{ represent single nodes} \\ \sum_{v \in i} \left[\frac{1}{|\Lambda_{vj}|} \sum_{\alpha \in \Lambda_{vj}} f^{\alpha}(v,j) \right] & \text{if } i \text{ represents a subgroup and not } j \\ \sum_{v \in i, u \in j} \left[\frac{1}{|\Lambda_{vu}|} \sum_{\alpha \in \Lambda_{vu}} f^{\alpha}(v,u) \right] & \text{if } i \text{ and } j \text{ represent subgroups} \end{cases} \tag{13}$$

$$w(i,j) = \begin{cases} |\Lambda_{ij}| & \text{if } i \text{ and } j \text{ represent single nodes} \\ \sum_{v \in i} |\Lambda_{vj}| & \text{if } i \text{ represents a subgroup and not } j \\ \sum_{v \in i, u \in j} |\Lambda_{vu}| & \text{if } i \text{ and } j \text{ represent subgroups} \end{cases} \tag{14}$$

Finally, we can run HodgeRank on each W_n, which will give us k rankings. We achieve our final ranking by stacking the groupings' rankings on top of each other according to their order from the naive rank.

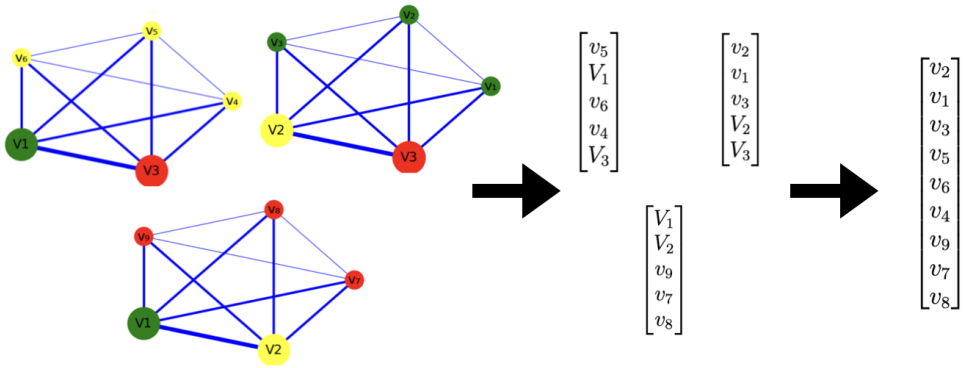
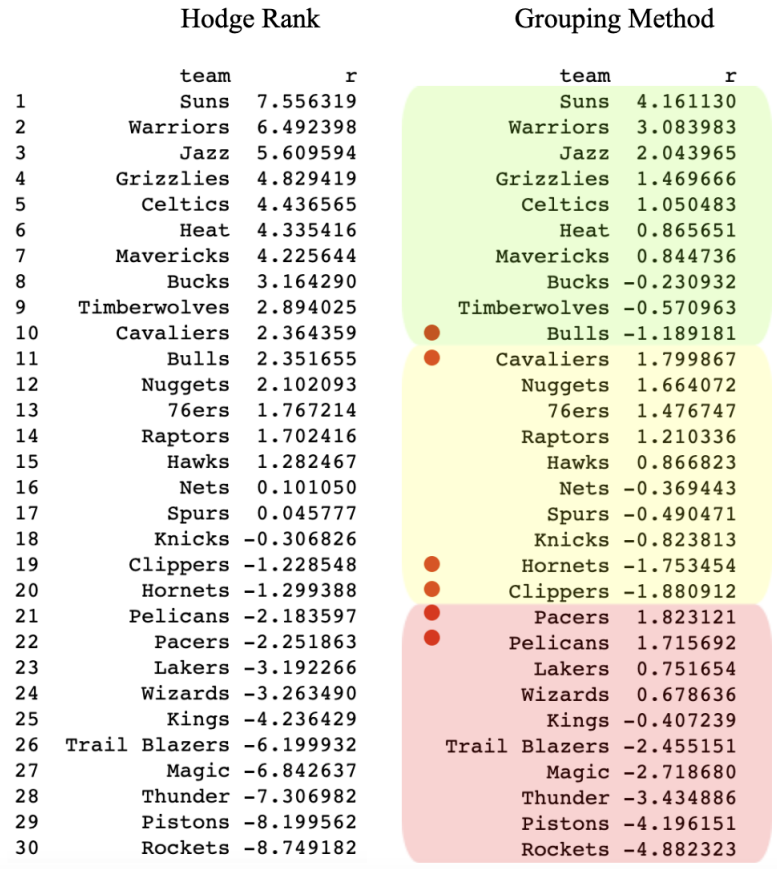


FIGURE 4. Universal ranking from grouping method. Note pseudo-nodes are omitted from the final ranking.

4.2. 2021 NBA season

Since the goal of the grouping method is to reduce the run time of HodgeRank, the efficacy of it should be measured by how well it approximates HodgeRank.

We process the NBA data set described in Section 3.2 through HodgeRank with groupings and compared the resulting ranking to the ranking without grouping. For this example we used $k = 3$.



	team	r
1	Suns	7.556319
2	Warriors	6.492398
3	Jazz	5.609594
4	Grizzlies	4.829419
5	Celtics	4.436565
6	Heat	4.335416
7	Mavericks	4.225644
8	Bucks	3.164290
9	Timberwolves	2.894025
10	Cavaliers	2.364359
11	Bulls	2.351655
12	Nuggets	2.102093
13	76ers	1.767214
14	Raptors	1.702416
15	Hawks	1.282467
16	Nets	0.101050
17	Spurs	0.045777
18	Knicks	-0.306826
19	Clippers	-1.228548
20	Hornets	-1.299388
21	Pelicans	-2.183597
22	Pacers	-2.251863
23	Lakers	-3.192266
24	Wizards	-3.263490
25	Kings	-4.236429
26	Trail Blazers	-6.199932
27	Magic	-6.842637
28	Thunder	-7.306982
29	Pistons	-8.199562
30	Rockets	-8.749182

	team	r
	Suns	4.161130
	Warriors	3.083983
	Jazz	2.043965
	Grizzlies	1.469666
	Celtics	1.050483
	Heat	0.865651
	Mavericks	0.844736
	Bucks	-0.230932
	Timberwolves	-0.570963
●	Bulls	-1.189181
●	Cavaliers	1.799867
	Nuggets	1.664072
	76ers	1.476747
	Raptors	1.210336
	Hawks	0.866823
	Nets	-0.369443
	Spurs	-0.490471
	Knicks	-0.823813
●	Hornets	-1.753454
●	Clippers	-1.880912
●	Pacers	1.823121
●	Pelicans	1.715692
	Lakers	0.751654
	Wizards	0.678636
	Kings	-0.407239
	Trail Blazers	-2.455151
	Magic	-2.718680
	Thunder	-3.434886
	Pistons	-4.196151
	Rockets	-4.882323

FIGURE 5. The three groups used to get the ranking on the right are outlined in different colors. Red dots indicate teams that were given a different ultimate ranking in the grouping method.

The ordinal association can be measured with Kendall's Tau, which is a statistic that takes in two ordered sets and returns a value between 0 and 1, where higher numbers symbolize greater correlation [13]. The Kendall's Tau of the two rankings was ≈ 0.959 implying the two are highly correlated. For reference, the Kendall's Tau of HodgeRank and the naive rank from Equation 12 is ≈ 0.931. Although the difference between the two isn't great, the results are still promising. It's possible that the grouping method may beat the naive rank by a greater margin in data sets with many more elements, as well as data sets where voter bias plays a greater role.

Something of interest is the mistaken ranking between the Bulls and Cavaliers. According to HodgeRank, the Cavaliers should have been ranked tenth, but in the grouping method they were ranked number 11. Even after applying HodgeRank,

the Cavaliers had no chance to make it to the tenth place, as it's group was designated to take the places 11-20. The same happened to the Bulls. Edge cases like these resulting from imperfections in the naive rank may direct future work.

4.3. Time complexity

In this section we analyze the time complexity of the method. Since the most computationally expensive step is the pseudo-inverse we'll only take a look at this operation. The complexity of computing the pseudo-inverses with HodgeRank is:

$$O(k(\frac{n}{k} + k - 1)^3) \tag{15}$$

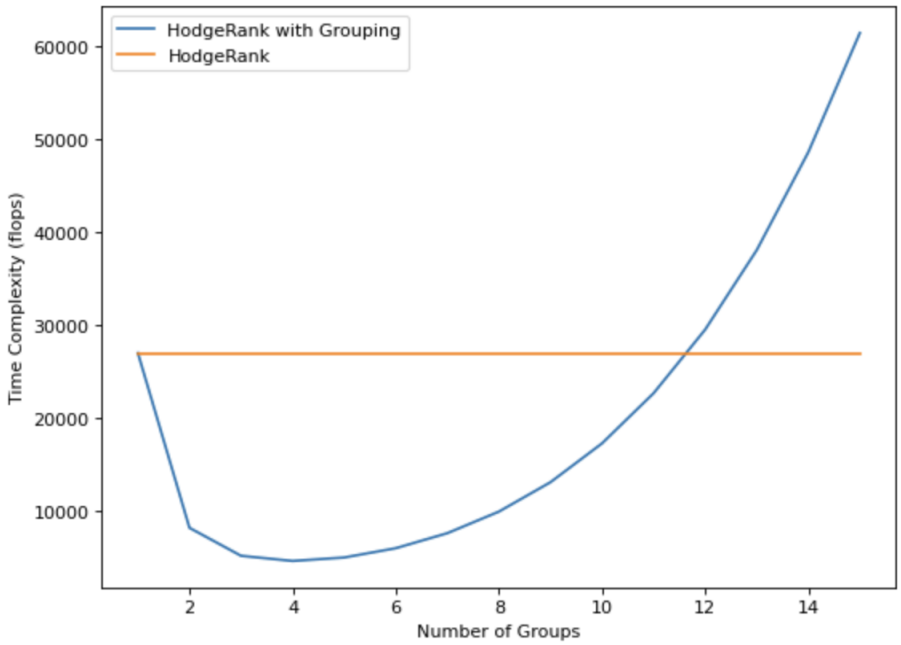


FIGURE 6. Time complexity of HodgeRank and HodgeRank with grouping on NBA data set, $n = 30$

As the graph depicts, there is an ideal range of k. As k becomes larger the number of times that the pseudo-inverse is taken is increased, resulting in the complexity of HodgeRank with grouping surpassing the complexity of normal HodgeRank. For $k = 1$, both algorithms are equivalent. Thus the ideal range for k is between 2 and 11 for $n = 30$. As it's a quite inexpensive operation, the best way we have found so far to find a suitable k has been to calculate the minimum of Expression 15 for any number of nodes n a data set may feature.

CHAPTER 5

Conclusion

HodgeRank is an algorithm which provides a ranking on data sets that may feature bias and incompleteness. It also offers a metric for judging the correctness of the ranking as well as a quantization of inconsistencies in the graph. We looked at two applications: running COVID-19 symptom data as well as NBA team data through the algorithm.

We proposed a new method to reduce the time complexity while maintaining the integrity of ranking. This method involves splitting the set into different groups to reduce the dimensions of the matrix being inverted. Measuring the performance of the grouping method on the NBA data set yielded a ranking that was highly correlated to the original HodgeRank. Since the grouping method is meant to address issues with data sets with many elements, it would be interesting to see its performance on data sets where the time complexity of the original method begins to affect the algorithm's utility. Analysis on the run time showed that picking a good number of groups is important to reduce runtime; for 30 nodes, the ideal number of groups was between 2 and 7, with values above 11 exhibiting a time complexity that was increasingly larger than that of HodgeRank.

Further work on the grouping method might address the issue of edge cases, where nodes that are sorted into the wrong tier at the start cause errors in the final ranking. Splitting up the groups in a more sophisticated way may address this and other issues. Additionally, it would be interesting to look into the performance and run time of embedded groupings for data sets with a great number of elements.

CHAPTER 6

Acknowledgements

I would like to thank my thesis advisor, Dr. Junyuan Lin, for her guidance and openness throughout this project. Working with her has been a privilege.

I would also like to thank Guangpeng Ren for his help on the project and advice on the writing of this thesis.

Additionally, I would like to give thanks to my academic advisor Dr. Alissa Crans, for guiding me through my mathematical endeavours.

Finally, I would like to express my deep appreciation for the math department's faculty, and for their mentorship in and out of the classroom.

Bibliography

[1] Nba games data. https://www.kaggle.com/datasets/nathanlauga/nba-games?select=games.csv%2Fversion1. Accessed: 2022-03-05.

[2] William W Cohen, Robert E Schapire, and Yoram Singer. Learning to order things. *Advances in neural information processing systems*, 10, 1997.

[3] Charles Colley, Junyuan Lin, Xiaozhe Hu, and Shuchin Aeron. Algebraic multigrid for least squares problems on graphs with applications to hodgerank. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 627–636, 2017.

[4] Wesley Colley. *Colley's bias free college football ranking method*. PhD thesis, Princeton University Princeton, NJ, USA, 2002.

[5] Koby Crammer and Yoram Singer. Pranking with ranking. *Advances in neural information processing systems*, 14, 2001.

[6] Herbert Aron David. *The method of paired comparisons*, volume 12. London, 1963.

[7] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[8] Petros Drineas and Michael W. Mahoney. Effective resistances, statistical leverage, and applications to linear equation solving. *CoRR*, abs/1005.3097, 2010.

[9] R.D. Falgout. An introduction to algebraic multigrid. *Computing in Science Engineering*, 8(6):24–33, 2006.

[10] Shelby Ferrier and Guangpeng Ren. hodgerank. https://github.com/shelby678/hodgerank/tree/main. Accessed: 2022-05-05.

[11] Lester R Ford Jr. Solution of a ranking problem from binary comparisons. *The American Mathematical Monthly*, 64(8P2):28–33, 1957.

[12] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

[13] Stephanie Glen. Kendall's tau (kendall rank correlation coefficient), Nov 2017.

[14] Louis Guttman. An approach for quantifying paired comparisons and rank order. *The Annals of Mathematical Statistics*, 17(2):144–163, 1946.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[16] Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.

[17] Mikio Kano and Akio Sakamoto. Ranking the vertices of a paired comparison digraph. *SIAM Journal on Algebraic Discrete Methods*, 6(1):79–92, 1985.

[18] Maurice G Kendall and B Babington Smith. On the method of paired comparisons. *Biometrika*, 31(3/4):324–345, 1940.

[19] Lek-Heng Lim. Hodge laplacians on graphs, 2015.

[20] David Nunan, Jon Brassey, Kamal Boyce, Antoni Gardner, and Maia Patrick-Smith. Covid-19 symptoms tracker, Jun 2020.
[21] The pandas development team. pandas-dev/pandas: Pandas, February 2020.
[22] Xue-Cheng Tai and Jinchao Xu. Global and uniform convergence of subspace correction methods for some convex optimization problems. *Math. Comput.*, 71(237):105–124, jan 2002.
[23] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.