Grouping Method for Faster Hodgerank Algorithms

Shelby Ferrier[a], Junyuan Lin[a,*], Guangpeng Ren[b]

[a]*Department of Mathematics, Loyola Marymount University, California, USA*
[b]*Institute of Mathematical Sciences, Claremont Graduate University, California, USA*

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. 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 integrity of ranking. We examine the efficacy and the time complexity of the proposed algorithm on the IMDb movie data set, with different selections on the number of groups.

In this study, our focus lies in the advancement of rapid algorithms for determining statistical rankings within highly incomplete and imbalanced

*Corresponding author.
Email addresses: shelbyferrier1@gmail.com (Shelby Ferrier), Junyuan.Lin@lmu.edu (Junyuan Lin), guangpeng.ren@cgu.edu (Guangpeng Ren)

[1]Declarations of interest: none
[2]This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors

datasets. Based on the HodgeRank algorithm, we describe the ranking problem on graphs. This allows us to formulate a least squares problem on graphs that gauges the reliability of the computed rankings. Given the escalating complexity of computations, as dataset dimensions amplify, our research is driven by the need for pragmatic solutions that accommodate the efficient and precise computation of rankings. Specifically, we group elements based on their tier in a naive ranking and run the HodgeRank algorithm on smaller subgroups, which makes the method faster while maintaining the integrity of the ranking. We examine the efficacy and the time complexity of the proposed algorithm on the IMDb movie data set and systematically explore the impact of varying group sizes on algorithmic performance. With the proposed grouping method, we are able to reduce the complexity of HodgeRank from $O(n^3)$ to $O(n^3/k^2)$, where k is the group size while safeguarding the ranking accuracy.

Keywords: statistical ranking, HodgeRank algorithm, approximation algorithm, Least Square Problem, Rank Biased Overlap

2020 MSC: 65K05, 90C59

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 [3; 5; 8; 7; 12; 10; 16; 9; 15]. Out of those methods, the HodgeRank algorithm, proposed by Jiang et al. [17], derives a global ranking from subjective, incomplete data sets which contain voters (e.g. people who have re-

viewed some movies) and scored elements (e.g. the ratings each person gives to a movie). The HodgeRank algorithm, distinguished from other ranking methods, analyzes pairwise rankings represented as edge flows on a graph using discrete or combinatorial Hodge theory. It takes 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 pairwise rankings in HodgeRank algorithm resolve this by focusing on the difference one voter gives between two different elements, as opposed to their individual ratings. 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 HodgeRank algorithm in Section 2.

The HodgeRank algorithm derives the statistical ranking problems into linear least squares problems on graphs and thus provides a measurement for the quality of the global ranking. With the relation to least squares problems on graphs, many mathematical solvers can be applied. As mentioned in [2], the Unsmoothed Aggregation Algebraic Multigrid (UA-AMG) [14] as a preconditioner for conjugate gradient (CG) yields efficient computation.

The HodgeRank algorithm is limited by its run time as the number of items being rated increases; we measured run times of more than an hour as the number increased over 2000, which we show in Section 4. This is caused by the costly step of computing the pseudo inverse of an $n \times n$ matrix where n is the number of elements to be ranked. The time complexity of this step is $O(n^3)$, thus to compute rankings over a very large set of elements (more

than 10,000) it is best to find a way to approximate the ranking, instead of using the original algorithm. We summarize some algorithms which do this, including the Algebraic Multigrid (AMG) method [2; 6] which cuts down on run time to $O(n \log n)$ while closely approximating the universal ranking.

In Section 3 we present a new method to cut down the time complexity of HodgeRank which sections the data into groups before computing the ranking. We tested the grouping method on IMDB movie rating data [1] and found the resulting ranking to be a strong approximation for the Hodge ranking. Additionally, we saw the accuracy of the results generally depending on how many groups were used, with less groups producing more accurate results. Finally we analyze the time complexity of the method and discuss the trade off between run time and accuracy when picking the best group size.

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. [17], 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. [17] and Colley et al. [2], 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:

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

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

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

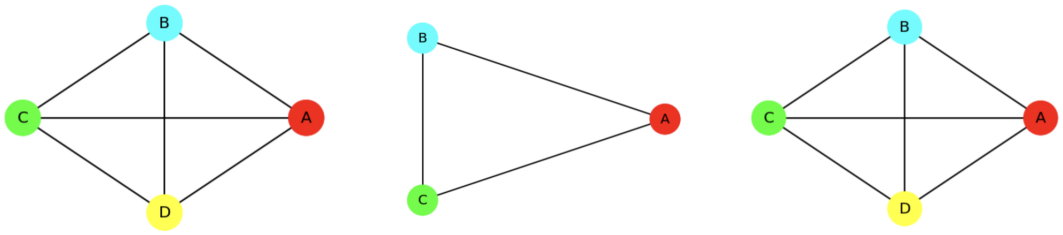
Also, we define the rankings as $R : \Lambda \times V \to \mathbb{R}$. For example, if voter α gave element i a score of 5, we would say $R(\alpha, i) = 5$.

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.

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

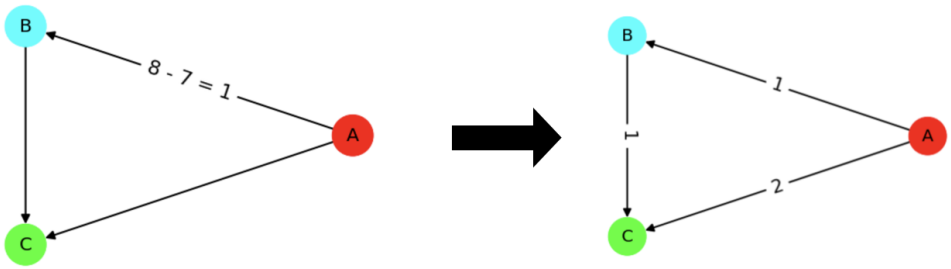


Figure 1: *

Patient 2's graph with pairwise comparisons

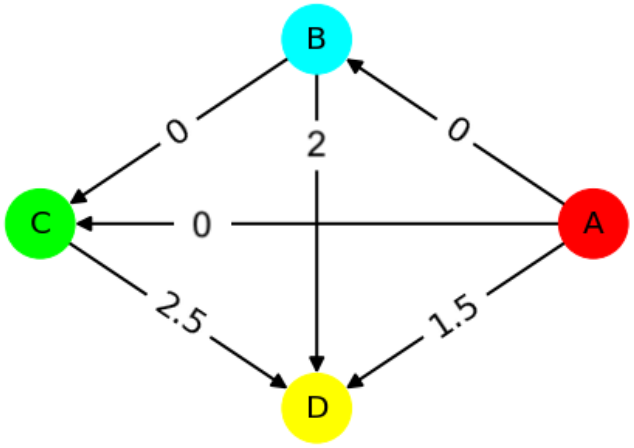
We can now define one graph, G, pertaining to all voter's data. G is a complete graph containing every alternative in V, so long as it's been voted on, as well as $\binom{|V|}{2}$ edges.

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, $f : V \times V \rightarrow$, 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 $\vec{f} \in^{|E|}$.

2.3. Least Squares Problem

Our goal is to find a universal rating $r : V \rightarrow$ 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.

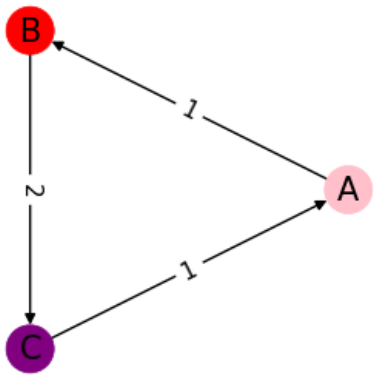
With this in mind we arrive at the function we use to judge the efficacy of any r:

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

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. [17] 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 $\vec{r} \in^{|V|}$ (where $\vec{r}$ is the vectorized form of r indexed by V) and local consistency $\vec{c} \in^{|T|}$ for any $\vec{f} \in^{|E|}$.

First we must define the boundary operators of the graph:

- The negative divergence, denoted by $\partial_1 :^{|E|} \rightarrow^{|V|}$, 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, $\partial_2 : ^{|T|} \rightarrow ^{|E|}$, 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 [2; 17; 11], we can show the following:

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

Therefore for any $\vec{f} \in ^{|E|}$ we can find $\vec{r} \in ^{|V|}$, $\vec{c} \in ^{|T|}$, 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 [11], with implementations demonstrated in [2; 17].

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:

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

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:

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

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 $\vec{x} \in^n$ 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. [6] which introduces the method, as well as in the paper by Colley et al. [2] which implements AMG directly with HodgeRank.

Tai et al. [18] 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. [13] [4] which employs a row sampling method to approximate large-scale matrix multiplication and reduce graph size.

We introduce a new method in Section 3 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.

3. 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.

3.1. Grouping Method

The first step in the grouping method is to obtain a naive ranking of elements, which we denote $r_0 : V \to$. 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.
$$r_0(i) = \frac{1}{|V|} \sum_{j \in V} f(j, i) \tag{11}$$

$$V = [v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9] \implies \begin{aligned} V_1 &= \{v_1, v_2, v_3\} \\ V_2 &= \{v_4, v_5, v_6\} \\ V_3 &= \{v_7, v_8, v_9\} \end{aligned}$$

Figure 2: Example: The elements in V which are indexed by their naive ranking are split into 3 subgroups

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

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 $k\binom{\lceil \frac{n}{k} \rceil}{2}$

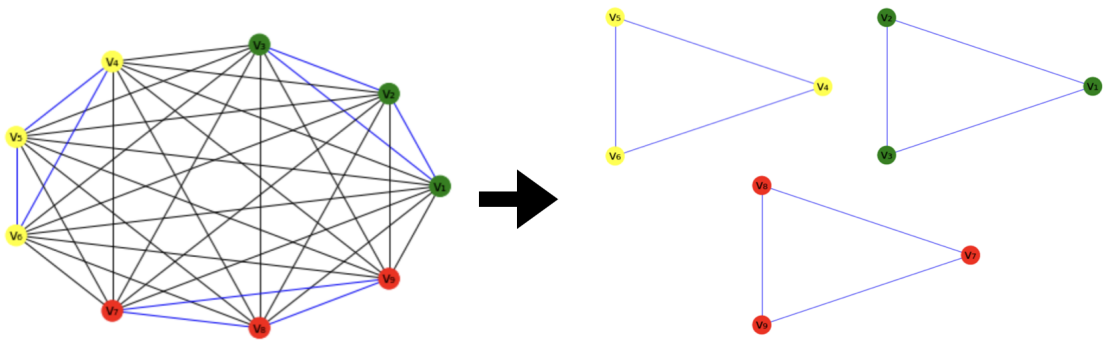


Figure 3: Running HodgeRank on smaller groups; in gray are all edges that are omitted

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.

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:

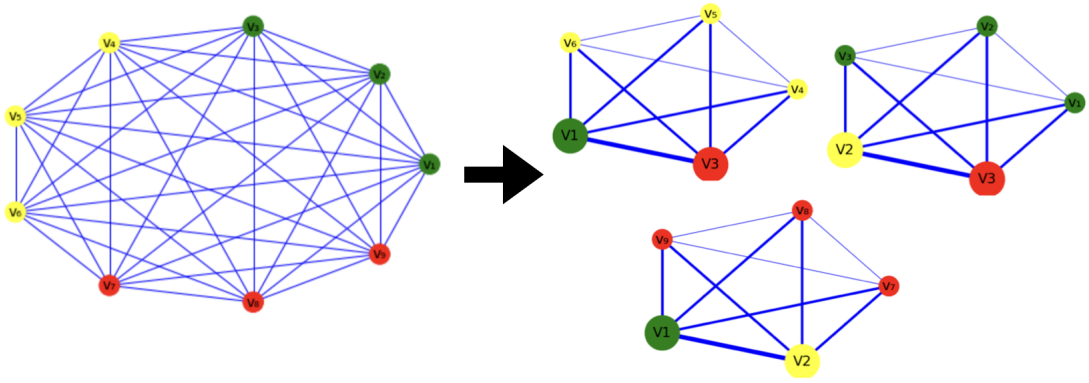


Figure 4: Groupings with pseudo-nodes; in this figure $V1$ represents the elements in the first group $\{v_1, v_2, v_3\}$, and so on. Edges on the right represent all edges between respective nodes, with thicker edges representing more edges from the original graph.

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

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

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.

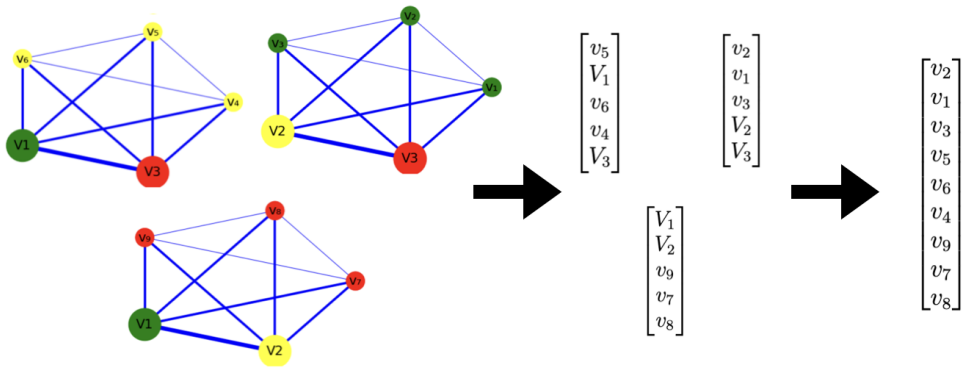


Figure 5: Universal ranking from grouping method. Note pseudo-nodes are omitted from the final ranking.

4. Results

To test the grouping method, we used the IMDb movie data set. The accuracy of a rating found using HodgeRank with grouping was measured by its similarity to the ranking found using the original HodgeRank algorithm. The metric we chose to measure this is the Rank Biased Overlap (RBO) which we detail later in the paper. Finally, from our experimentation we discuss how to pick the best group size by balancing accuracy and run time.

4.1. Data

For our experimentation, we used the IMDb movie data set, which is a collection of IMDb movies updated on 27 December 2020 including their

UserID, MovieID and ratings collected by Vahid Baghi and uploaded to IEEE Dataport.

The dataset offers 4,669,820 ratings from 1,499,238 users to 351,109 movies.

4.1.1. Preprocessing

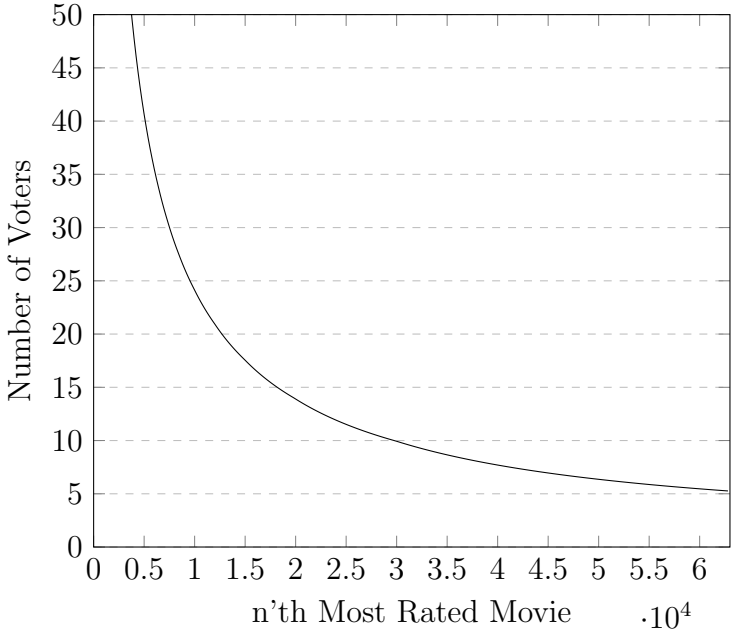
We take a subset of 330,945 from 500,000 ratings data points by filtering movies with the condition of at least two users rating a movie. Then, we import it into MATLAB, sparse the matrix, and process the matrix to L, which is the graph Laplacian of the network flow, and b, which is the the right hand side from $\partial_1 \cdot f$, and the incidence matrix. We also sample different graph Laplacians corresponding to different divides of MovieID ranged from 1000 to 20,000 and only maintain the largest connected component from the incidence matrix as a complete graph. Thus, we can obtain the most representative edge flow results. Finally, we produce the ranking r by operating $L\backslash b$ in MATLAB.

4.1.2. Organizing by Popularity

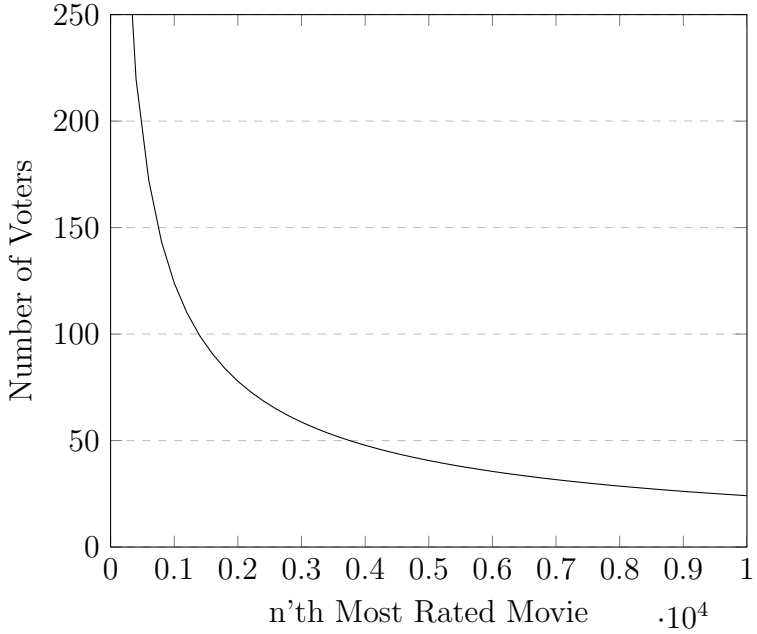
The largest connected component in the IMDb movie data set contains a total of 62,917 movies, including 29,945 movies that were rated by at least two people. We chose to analyze the n most rated movies, because we wanted to run the algorithms on data sets which had the most data possible. Upon running the grouping method on the IMDb data set, we saw greatly decreased accuracy upon running it with the top 10,000 most popular movies compared to the top 5,000 movies. We suspect this

is because the data set with 10,000 movies in it had fewer rankings on average than that with 5,000 movies. Below is a breakdown of the number of voters per the n'th most popular movie.

Average Number of Voters for First n Most Rated Movies (all movies)



Average Number of Voters For First N Most Rated Movies (Top 10,000 Movies)



The least rated movie in the trial with 5,000 movies had r ratings. Thus, we recommend that each item in a data set used for the grouping method has at least r ratings, to conserve accuracy. Note that a rating only counts if it came from a voter who has rated at least one other item in the data set, or else it won't affect the final ranking.

4.2. Computing Environment

The numerical tests are conducted with a 1.80 GHz Intel Core i7-8550U CPU, a quad-core processor, and 16 GB of RAM.

4.3. RBO

We decided to use Rank-Biased Overlap(RBO) method [19] to compare our different ranked lists. Since RBO is a similarity measure, theoretically it should perform better for our tasks than Kendall Tau which is a distance measure. RBO is using weights for each rank position which are derived from a convergent series. The goals of RBO are to handle non-conjointness, weight high ranks more heavily than low, and be monotonic with increasing depth of evaluation.

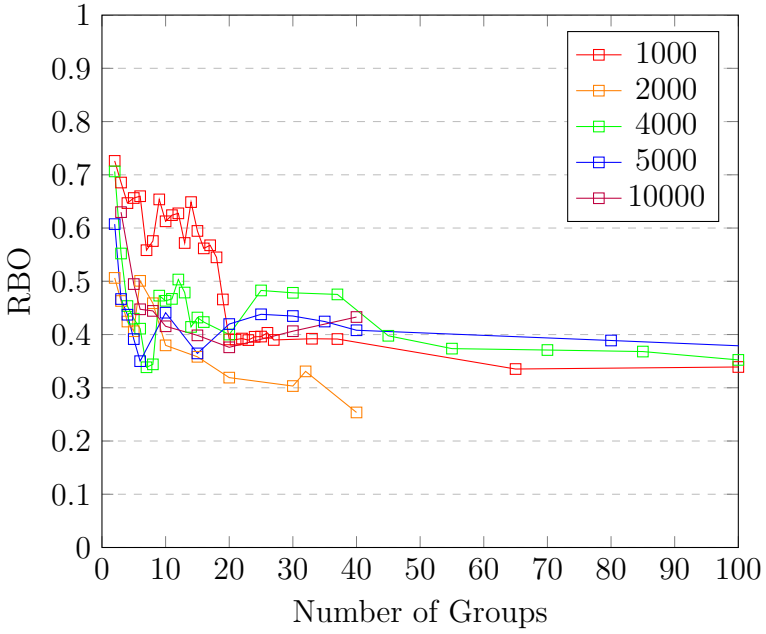
$$RBO(S, T, p) = (1 - p) \sum_{d=1}^{\infty} p^{d-1} A_d$$

S and T are two distinct ranked lists. d is the depth of the list. A_d is the agreement between S and T by the proportion of the overlap size over the depth, essentially $\frac{X_d}{d}$. The parameter p represents the steepness of decline in weights. For example, a smaller p indicates more top-weighted metric. The RBO score falls into the interval of [0, 1], where 0 means disjoint and 1 means identical.

4.4. Accuracy

To test accuracy, we compared the ranking found from using grouping with HodgeRank using Rank-Biased Overlap (RBO). A score of 0 implies no correlation and is what you'd see if you compared to randomly organized rankings. A score of 1 means perfect correlation.

The effect of number of groups on ranking accuracy



The graph shows accurate results for very few groups for the trials with 1000, 2000, 4000, and 5000 movies. In comparison many trials with 10,000 movies show RBO scores that are close to 0, implying an inaccurate ranking. We predict this is because the group with 10,000 movies has more movies that were rated by less people than the other sets. This method depends on comparing ratings between voters, so if there are not many ratings on a movie the overall rating will not be as accurate.

Overall, the rankings found using the grouping method show high correlation with the ranking found using just HodgeRank. As we'll see next, the run time is greatly reduced by using groupings.

4.5. Run Time

First we'll calculate the theoretical time complexity for this method and recommend a general formula for picking the best number of groups.

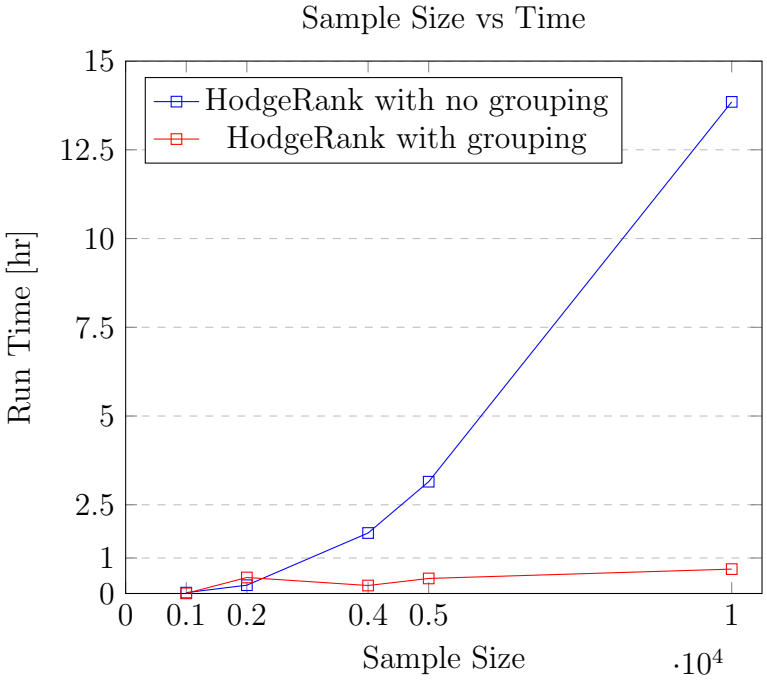
Since the most computationally expensive step in both algorithms is taking the pseudo-inverse, we focus on this operation. The complexity of computing the pseudo-inverses during HodgeRank with grouping is:

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

From our experimentation, we found that using the fewest groups gives the most accurate ranking. Thus, we recommend using as few groups as possible, with two groups being ideal, while maintaining a sufficiently small run time. As the number of items being ranked increases, it may not be feasible to compute the universal ranking with two groups, thus we generally recommend using $k = log_{21}(n)$ where k is the number of groups. Plugging our recommended k into equation 15, the run time drops from $O(n^3)$ for Hodgerank without grouping to $O(\frac{n^3}{log_{21}(n)})$.

Using the IMDb movie rating data set, we obtained the following run times. All run times are measured on trials where $k = 2$, which gave the most accurate results.

As the graph shows, the run times of all trials which used the grouping method took less than an hour to run, whereas the longest run time for the original HodgeRank algorithm is 13.85 hours, for 10,000 movies.



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.

References

[1] Vahid Baghi. Imdb users' ratings dataset, 2020.

[2] X. Hu S. Aeron C. Colley, J. Lin. 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.

[3] W. Colley. *Colleyâs bias free college football ranking method*. PhD thesis, Princeton University Princeton, NJ, USA, 2002.

[4] M.W. Mahoney D. Petros. Effective resistances, statistical leverage, and applications to linear equation solving. *CoRR*, abs/1005.3097, 2010.

[5] H.A. David. *The method of paired comparisons*, volume 12. London, 1963.

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

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

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

[9] Y. Singer K. Crammer. Pranking with ranking. *Advances in neural information processing systems*, 14, 2001.

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

[11] L.H. Lim. Hodge laplacians on graphs, 2015.

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

[13] M.W. Mahoney P. Drineas, R. Kannan. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[14] Klaus Stüben. A review of algebraic multigrid. *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359, 2001.

[15] J. Zobel W. Webber, A. Moffat. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

[16] Y. Singer W.W. Cohen, R.E. Schapire. Learning to order things. *Advances in neural information processing systems*, 10, 1997.

[17] Y. Yao Y. Ye X. Jiang, L.H. Lim. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.

[18] J. Xu X.C. Tai. Global and uniform convergence of subspace correction methods for some convex optimization problems. *Math. Comput.*, 71(237):105â124, jan 2002.

[19] R.E. Schapire Y. Singer Y. Freund, R. Iyer. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems*, 28(Nov):1–38, 2010.