Faculty of Engineering and Science
Aalborg University

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science

TITLE:
Beyond Individual Recommendation - Aggregation Methods for Group Based Recommender Systems

PROJECT PERIOD:
1st February 2017 -
13th June 2017

PROJECT GROUP:
mi1012f17

GROUP MEMBERS:
Lasse Drustrup Christensen
Lukas Nic Dalgaard

SUPERVISOR:
Peter Dolog

PAPER PAGES: 9
APPENDIX PAGES: 2
TOTAL PAGES: 11

ABSTRACT:

In this paper we evaluate four different aggregation methods, Borda Count, Markov Chain, Spearman's Footrule, and Average, on four different measures, nDCG using ranks, nDCG using ratings, Kendall Tau Distance, and Spearman's Footrule Distance.
For individual recommendation, we use SVD++ from MyMediaLite, and groups generated from the MovieLens 100K dataset, of sizes ranging from 4 to 40.
Our findings show that Borda Count has the overall best performance. Markov Chain, using the Copeland method as a heuristic, also nearly performs on par with Borda Count, and that the quality of the recommendations drop as the size of groups increase per all measures, but that the decrease becomes almost nothing after group size 20.

Summary

Most commonly in recommendation, it is for a single person. The classic problem for the Recommendation System is to provide the best item from a variety of options to a single user. It has branched OUT into a multitude of branches such as collaborative and content-based filtering.

Today, recommender systems concern themselves about where we should eat, what music to listen to, what movie to watch, or where our next vacation should go to.

However, none of the scenarios above are uniquely activities done alone. Some are traditionally outright viewed as group activities for most people by default. As such, the concept of a group recommender is an intuitive extension to the traditional recommender.

This article deals with recommendation for groups of people. The problem is reflected in many other aspects of life and it is radically different from the challenges of a normal recommender system. Voting shares similarities with the challenges seen in group recommendation, as the challenge is to recommend the option that is the least opposed by all parties or satisfies some other criteria for approval in the group.

So instead of a recommendation, the challenge is counting votes. For Group Recommendation, this is usually defined as aggregation, and many aggregation methods exist and are used for many domains. Borda Count, which exists both as a voting and aggregation method, is one such example and is used in this master thesis project.

Borda Count in particular was notable for the project. In the previous semester, the group working on this project was exploring extensions of Borda Count. The results were promising, but there was a lack of a ground truth to really give meaning to the results.

For the project, initially, we were chasing the possibility of creating a dataset to establish a ground truth for our earlier findings. The dataset itself would have been a great contribution as there is not a lot of available data for the group recommendation field. However, given the amount of data needed for a proper dataset, we had to look towards paid services to attract the numbers needed. In this case, we turned to Amazon Mechanical Turk, where it is possible to pay people to answer surveys or other simple tasks. With that in mind we applied for funding and got a 100 euro to spend for the project with the stipulations that the dataset be freely available to all AAU students and were reusable for others in the field.

So to evaluate our results, we could not ask amazon turkers to rate our group recommendations, as they would not be reusable. As such the plan for the survey was to have the users provide the recommendations on the assumption that humans are good group recommender systems. We could then compare group recommender systems in how they recommended as opposed to humans.

As the survey designer for Mechanical Turk was not complex enough to cover our use case, it was decided to make our own server and webpage to handle the survey and collect the answers. Initially, we started development on a Java server using JavaServer Pages, but after a while we switched to python with the Django framework. For webhosting, we found a provider online.

The survey was simple. The survey participant was given information about the preferences for a group, and was asked to make a ranked list of recommendations for the group.

To avoid overloading the participant with information and make the survey take too long, we spent a lot of time on making it easier for the participant.

However, 3 days after we launched the survey, we found ourselves suspended from the Mechanical Turk with no recourse for recouping the money or refuting the suspension.

At this point, around half the alloted time for the project was gone and we had nowhere near enough data to establish any ground truth, and we turned the project towards testing new aggregation methods in the group recommendation domain. Additionally, we would use many types of measures to make up for our lack of a real dataset.

In the end, we implemented many aggregation methods, of which Borda Count, a Markov Chain variant; MC_4, Spearman's footrule, and Average made it to the paper.

A paper on these methods had made some interesting insights for these measures on group sizes between 2 and 8, so we pivoted to further test the results of the paper. With the extra measures we implemented, we also confirmed the results for more than just Normalized Discounted Cumulative Gain.

Beyond Individual Recommendation

Aggregation Methods for Group Based Recommender Systems

Lasse Drustrup Christensen
Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
Aalborg East, Denmark 9220
ldch11@student.aau.dk

Lukas Nic Dalgaard
Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
Aalborg East, Denmark 9220
lnda12@student.aau.dk

Abstract

In this paper we evaluate four different aggregation methods, Borda Count, Markov Chain, Spearman's Footrule, and Average, on four different measures, nDCG using ranks, nDCG using ratings, Kendall Tau Distance, and Spearman's Footrule Distance.

For individual recommendation, we use SVD++ from MyMediaLite, and groups generated from the MovieLens 100K dataset, of sizes ranging from 4 to 40.

Our findings show that Borda Count has the overall best performance. Markov Chain, using the Copeland method as a heuristic, also nearly performs on par with Borda Count, and that the quality of the recommendations drop as the size of groups increase per all measures, but that the decrease becomes almost nothing after group size 20.

Keywords Group Recommendation, Rank Aggregation, Borda Count, Markov Chain, Spearman's Footrule, Average, nDCG, Kendall Tau Distance, Spearman's Footrule Distance

1 Introduction

Many of the decisions we make are based on recommendations, from either people we know or recommender systems tailored to personal preferences. This can be helpful due to the high amounts of information we process in our everyday lives[5]. The recommendations, or more specifically in our case, the recommender system, can cut down the number of options to a manageable level and thereby augment the decision-making process without forcing a decision.

The problem with the traditional recommender systems is that they typically make recommendations tailored to one person but often these decisions needs to be taken in a social context.

For some scenarios, such as for selecting a movie on a streaming service, finding a restaurant, or deciding on a vacation destination, the inclusion of a social context would change the problem from that of knowing ones own preferences to that of an entire group in the given context.

A problem regarding taking the social context into consideration is that the recommender has to strive for consensus between the people it recommends to. An already complex problem is made even harder by having to solve it for multiple users simultaneously with new rules in play. From here, we will reference to this problem as making a group recommendation.

When making group recommendations there are two main approaches, namely profile aggregation and recommendation aggregtion [2]. The idea behind profile aggregation is to aggregate the users' preferences into a single group profile and make aggregations based on that profile. The other approach is to consider each user individually and aggregate the recommendations for the users into one aggregation that fits the groups preferences. In this paper we have chosen to focus on aggregation recommendation.

As we are going to aggregate the users recommendations we have chosen to only focus on the top-k part of their recommendations and return a list of recommendations of size k as a result. Furthermore, the top-k lists will be ranked with the highest rated item at first position on the list.

With ranked top-k lists being partial lists, we have selected four types of aggregation methods which have shown good results when used for aggregating partial lists. The methods we used were Borda Count, Markov Chain, Spearman's Footrule, and Average[4, 11].

For group recommendations we faced the challenge of evaluating the result without a dataset to provide a ground truth. However, from the information retrieval domain, we found measures to evaluate the quality of queries that can be used to evaluate the quality of a ranked list of recommendations and there are many datasets available for individual recommendations.

One such dataset is the 100k MovieLens dataset used by Baltrunas et al for a group recommender setup[1, 8]. They used aggregation methods such as Borda Count and Average and evaluated their performance using Normalized Discounted Cumulative Gain. Their tests were done on groups of size 2, 3, 4, and 8 and the findings they made were that the quality of the recommendation did not always drop even when the group size grew. Furthermore, their result showed a significant quality drop from group size 4 to 8. We adapted some of their approaches, more specifically Borda Count, Average, and Normalized Discounted Cumulative Gain, and setup in order to make further tests for larger groups to document if the decrease continues at the same rate as between group size 4 and 8.

1.1 Research Questions

Among common aggregation methods, given ranked top-k lists $\tau_1, ..., \tau_u$, where u is the number of group members, which method can provide the most optimal group recommendation per measures such as satisfaction or distance from individual preferences of the group?

Baltrunas et al supplies us with some results for the performance of a group recommender setup. Their results for group sizes 2, 3, and 4 are very close and perform well, but they show a drop off in performance for group size 4 to 8. Is it possible to reproduce similar test results and with larger groups to investigate if the drop off continues? Furthermore, is it possible to verify the results with additional measures?

1.2 Structure of the Paper

The structure of the paper is as follows. Section 2 gives a short overview of the implemented system. Section 3 describes the methods used of making a group recommendation. In Section 4 we will present the evaluation including setup and our results. In Section 5 we discuss the results of the evaluation and in Section 6 we will present our conclusion and future work.

2 System Overview

In this section we give a short overview of the group recommendation system which is depicted in Figure 1. Each of the stages will be outlined with a short description.

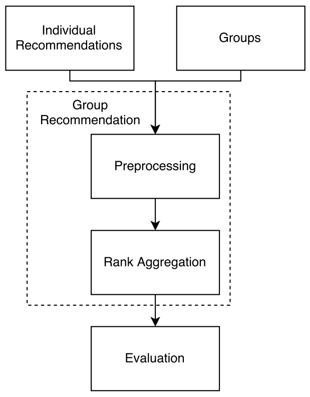


Figure 1. Stages of the group recommender system

Individual Recommendation We make individual recommendations for every user. The recommendation methods used in this step is interchangeable and can be selected to fit the data and purpose of the recommendation. The only condition for the recommender is that it finds a complete list of recommendations of the users in a group. In Section 4.1.2 the recommender we use is further elaborated on.

Groups In this stage we generated a list of groups for testing purposes. These consist of user id's and were generated at random but it is ensured that the same user only appears once in each group. The specific setup of the groups is described in Section 4.1.3.

Group Recommendation The group recommendation part consists of two stages, namely preprocessing and rank aggregation.

- Preprocessing is needed to find the individual recommendations belonging to the users in a certain group and format them for the rank aggregation.
- Rank aggregation combines the individual recommendations into a list of size k which should represent the groups preferences.

A more detailed description of the stages in group recommendation can be found in Section 3.

Evaluation The last stage is evaluation. In this stage several tests and measurements are performed. The setup and results of this stage is shown in Section 4.

3 Group Recommendation

This section documents the preprocessing done and outlines the rank aggregation methods used in order make the recommendation aggregation into a group recommendation.

3.1 Preprocessing

Prior to making the rank aggregation we do some preprocessing. As specified in Figure 1 the preprocessing stages get groups and all the individual recommendations as input. Preprocessing is concerned with constructing a top-k list for each of the users in a specific group based on the individual recommendations.

A top-k list is specified as a ranked list of length k consisting of the highest rated items order in descending order. More specifically let τ be a top-k list and let $\tau(i)$ be the rating of item i, which is an arbitrary item, then list is ranked if $\tau(1) > \tau(2) > \ldots > \tau(k)$.

The top-k lists are stored in an array which is used as input for the aggregation methods.

3.2 Rank Aggregation

In this section we describe the aggregation methods. Common for all the methods is that they aggregate an array of top-k lists into one ranked list, ω, of length k containing recommendations of a group. The order of ω may differ between the methods as they rank it based on which items they deem most relevant for the group, with the most relevant item first.

3.2.1 Borda Count

Borda Count(BC) was originally used as a voting system but has over the years been used in different domains because of its ability to aggregate ranked lists[1, 11].

The way BC works as a voting system is by the voters ranking the k candidates by assigning votes 1 to k, giving k points to their favorite candidate $k-1$ to their second favorite down to 1 point to their least favorite.

In our case we feed the BC method with an array of top-k lists, the items in the lists are assigned points by giving item one k points down to 1 point for item k[3]. Naturally, an item not on a users' top-k scores zero points from that user. The way the aggregations are made is by using Equation 1. U is the set of users top-k lists in a group and τ_u is a users' list. I is the set of items given by the union of all lists in U, so $I = \tau_1 \cup \ldots \cup \tau_u$ and i is an item in I. The equation, assuming that $\tau_u(i)$ is the points of item i in a top-k list, sums of all items $i \in I$ the points of that item from each of the users' top-k lists.

$$bc(i) = \sum_{u \in U} \tau_u(i) \tag{1}$$

The k items getting most points is returned as the recommendation list, ω, which is in descending order.

3.2.2 Markov Chain

Dwork et al propose a Markov Chain for aggregating ranked lists, called MC$_4$ [4]. MC$_4$ generalizes the heuristics of the Copeland Method, where a winner is the candidate which wins the most pairwise contests[15].

MC$_4$ is a process where we note the possibility of transitioning from one state to another state over time. The MC$_4$ state space, S, corresponds to a set of all the items, I, such that $S = \{1, 2, \ldots, |I|\}$. The transition probabilities between states are represented by a transition matrix, $P = |I| \times |I|$, covering the probability, p_{ij}, between any item pair $i \in I$ and $j \in I$.

To calculate the probabilities, let c_i be the set of items from I, that for the majority of the ranked lists we aggregate for, $\tau_1, \tau_2, ..., \tau_u$, it holds that $\tau_u(i) > \tau_u(j)$. As such, for the item placed first on every ranked list, $c_i = I - i$, and for the item placed last for every list would have c_i be the empty set. The probabilities of P is found according to Equation 2. λ is a variable for teleporting that makes P irreducible such that it has no absorbing states, and provides a small increase in accuracy. Via tuning, we found that $\lambda = 0.05$ is a good value. For the case of a missing item from either or both lists, the item is considered to be on the lowest possible rank.

$$p_{ij} = (\frac{|c_i|}{|I|})(1 - \lambda) + (\frac{\lambda}{|I|}) \tag{2}$$

For the probability of state i staying in state i, we have Equation 3.

$$p_{ii} = (\frac{|I| - |c_i|}{|I|})(1 - \lambda) + (\frac{\lambda}{|I|}) \tag{3}$$

When the transition matrix is calculated, the result can be found via the stationary distribution for P. A distribution vector is a vector of size $|I|$, which holds non-negative values, representing how the states are distributed. For an initial distribution, x, then xP^t, is the same initial distribution after t steps down the chain. The stationary distribution is where the state distribution stops changing regardless of taking more steps.

For practical purposes, we can approximate the stationary distribution for P via application of the power-iteration algorithm. So the approximate distribution, r, is found in Equation 4 for a number of steps, t. Via tuning, we found that $t = 30$ was a good value.

$$r = xP^t \tag{4}$$

The result of MC$_4$, ω, is then found as the k items with the biggest shares of r.

3.2.3 Spearman's Footrule

Dwork et al propose Spearman's footrule(SF) for aggregating ranked lists[4]. SF utilizes bipartite graphs from graph theory to construct a weighted complete bipartite graph (I, P, W). Let I be the set of items equal to the union of the top-k lists $\tau_1, ..., \tau_u$, where u is the number of users in a group. Then we have the set $P = \{1, ..., k\}$, which are the available positions in the list to be recommended. Lastly, the set W is the set of edge weights between items $i \in I$ and positions $p \in P$. The weights $W(i, p)$ are found by using the scaled footrule distance equation which can be seen in Equation 5[4].

$$W(i, p) = \sum_{n=1}^{k} |\frac{\tau_n(i)}{k} - \frac{p}{k}| \tag{5}$$

As we work with partial lists we will encounter lists with missing items. For this reason we have added a second case in addition to the approach described by Dwork et al, which can be seen in Equation 6. In this case we adapt the variable ℓ from Spearman's footrule distance. This variable is used on partial lists for measuring distance. ℓ needs to be larger than k and in our case it is $k + 1$. The reason for this is to punish infrequent items by giving them a higher weight.

$$W(i, p) = \sum_{n=1}^{k} |(\frac{\ell}{k} - \frac{p}{k}| \tag{6}$$

After determining the edge weights, the problem can be solved as a minimum cost maximum matching problem, which is the problem of finding the highest number of node matches with the lowest edge cost. To do this, we decided to use the Munkres extension of the Hungarian method[12]. The result of this method, ω, is a ranked list of size k containing the recommended items.

3.2.4 Average

We choose Average(Avg) aggregation as it is one of the more commonly used and well performing methods within group recommendation[14]. Baltrunas et al, also used an Average aggregation method as one of their measures for their setup with ratings[1]. Our implementation only considers the items in the top-k list. It finds the union of all the users' top-k lists, $u \in U$, so $I = \tau_1 \cup ... \cup \tau_u$. The Avg method then uses the full lists, $\sigma_1, ..., \sigma_u$, from the individual recommendations to find the average rating for the items $i \in I$. Equation 7 illustrates how Avg works.

$$Avg(i) = \sum_{u \in U} \frac{\sigma_u(i)}{|U|} \tag{7}$$

4 Evaluation

In this section we show our evaluation of the aggregation methods described in Section 3.2. The aim of the evaluation was finding an appropriate aggregation strategy for group recommender systems according to the 4 measures. Second to that was to confirm earlier findings on group recommender systems and extend them to other measures and bigger group sizes.

4.1 Setup

Throughout these tests we have decided to assign k the size 10. Figure 2 show the basic setup of the test leading to the evaluation. The lists of individual recommendations for a group of size u will be put through an aggregation method before outputting a list of k ranked items. The input is made through a combination of the available data, individual recommendations, and the group generation.

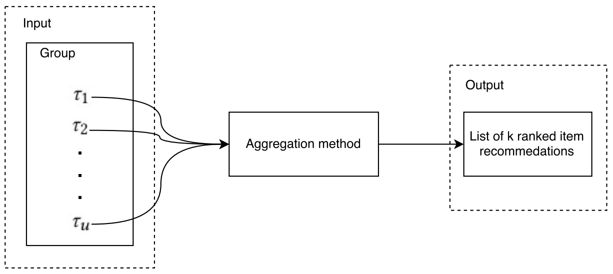


Figure 2. Concept of the test setup. Aggregation methods take in top-k lists and returns a list of recommendations.

4.1.1 Dataset

We used the MovieLens 100k dataset published by GroupLens in 1998[8]. MovieLens 100K contains 100.000 ratings between 1 to 5 collected from 943 users across 1682 movies. With room for approximately one and a half million ratings, the 100k rating dataset is sparse.

4.1.2 Individual Recommendations

For rating prediction, we used the library called MyMediaLite[7]. MyMediaLite is a library for .NET that holds a bundle of recommendation methods for both item recommendation and rating prediction. We will be using the library, because this gives a tested foundation that is easy to reproduce and the focus of our paper lies in testing the aggregation methods.

Among the methods provided by MyMediaLite, SVD++ is one of the best performing on the 100k dataset on their own records using the parameters in Table 1[1]. For the sake of convenience we are using the same parameters as they are proven to be efficient.

Latent Factors	50
Regularization	1
Bias Regularization	0.005
Learning Rate	0.01
Bias Learning Rate	0.07
Number of iterations	50
Frequency Regularization	True

Table 1. Parameters values for the SVD++ component

4.1.3 Group Generation

For the aggregation we made groups consisting of 4, 8, 12, 16, 20, and 40 users from the MovieLens 100K dataset. The reason for this is because we wanted to reproduce and futher the results found by Baltrunas et al[1], who had group sizes from 2 to 8.

Given that the dataset contains 943 users, we limited our group size to 40, as to not have any groups containing more than 5% of all the users. This ensured some amount of diversity in the groups. 40 is also ten times the size of our smallest group, enough to indicate the trend for the quality of recommendations. The groups were created of randomly picked users, and the same user can appear in multiple groups, but never in the same group twice.

4.1.4 Satisfaction Measures

We measure the groups' satisfaction of a recommended list, ω, according to Normalized Discounted Cumulative Gain(nDCG). We used two different variations which are described in this section. nDCG is used for measuring the quality of ranked lists against user preferences, which is commonly used within the information retrieval field for comparing ranked lists of queries[9].

Normalized Discounted Cumulative Gain

For evaluating the quality of the result list, ω, we use nDCG by comparing it against users' top-k lists τ.

In Equation 8, a *DCG* value is calculated for a set of k ranked items as the sum of the set of items' relevance scores divided by the logarithm of its ranking $n+1$ where $n \geq 1$. The relevance, rel, is defined as a set of scores for items in the ω, compared to the position of the items in a correspondence τ list. More specifically for all items $i \in \omega$, if $i \in \tau$ then, assuming that $\tau(i)$ and $\omega(i)$ is the position of i in the lists, $rel(\omega(i)) = \tau(i)$. If $i \notin \tau$, then $rel(\omega(i)) = 0$.

[1]www.mymedialite.net/examples/datasets.html

$$\text{DCG}_k = \sum_{n=1}^{k} \frac{rel(n)}{\log_2(n+1)} \tag{8}$$

$$\text{nDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k} \tag{9}$$

In Equation 9, the *DCG* value is normalized against the ideal *DCG*, *IDCG*, which is the *DCG* value based on the ideal recommendations for that user. The *IDCG* is the set *ideal*, which in this case, as we are concerned with the position of the items, is $k, \ldots, 1$ for every top-k list, exchanged with rel in Equation 8.

Rating nDCG

In an effort to more accurately portray the quality of the ranking, we present the Rating nDCG measure.

The difference is that the relevance score of items are not represented by their ranking, but directly from the predicted ratings for that item with the set of rating values, rat. It is defined as the rating for every $i \in \omega$ for that item in a users full list of recommendations σ, such that $\omega(i)$ is the position of i in ω and $\sigma(i)$ is the rating of item i for a user, and $rat(\omega(i) = \sigma(i)$. Intuitively when the relevance set is changed to a rating set for *DCG* it also needs to be changed for *IDCG*. The *ideal* set for Rating nDCG is the ratings of the items on a top-k list, τ. So if $\tau(1)$ is the rating of item one, then the *ideal* set is $\tau(1), \ldots, \tau(k)$.

This slightly changes the DCG calcuation for Rating nDCG into Equation 10.

$$\text{RatingDCG}_k = \sum_{n=1}^{k} \frac{rat(n)}{\log_2(n+1)} \tag{10}$$

As there is often not much overlap between individual users' recommendations, it can better reflect the quality of a recommendation for that user, as a recommendation is not punished as harshly by not including items on a user's individual top-k list. It also more closely reflects the user's rating of an item's relevance, as it is not decided by the ranking. Overall, this leads to much higher nDCG scores, as even total misses are no longer necessarily seen as such.

Conversely, it can be argued Rating nDCG measure is inferior to nDCG, as the aggregation methods, aside from Avg, only consider the ranked elements when making the aggregation, so the increased score is, from the perspective of the aggregation methods, entirely random.

4.1.5 Distance Measures

Before going through the distance measures, we want to cover some general notations that both methods use. $\tau(i)$ and $\omega(i)$ is the notation for the position of item i in τ and ω. $Z = \tau \cap \omega$, $z = |Z|$, S is the set of items only in τ and not in ω and T is the set of items in τ not in ω. k is the length of the top-k lists.

Kendall Tau Distance

The idea of Kendall tau distance(KTD) is to compare two ranked lists based on the order in which the items appear[6]. This means that it makes pairwise comparisons of item indexes $\{i, j\}$ where $i < j$, so that if i is before j in τ then this should also be the case in ω in order to get a good score. The score is based on a count of how many times i and j are in reverse order. In Equation 11, KDT is outlined. P is the set of unordered pairs of distinct items in τ and ω. If i and j is in the same order in τ and ω then $\bar{K}_{i,j}(\tau, \omega) = 0$ but if i and j is in reverse order then $\bar{K}_{i,j}(\tau, \omega) = 1$.

$$K(\tau,\omega) = \sum_{\{i,j\}\in P} \bar{K}_{i,j}(\tau,\omega) \quad (11)$$

In order to adjust KTD for partial lists we used the K_{Haus} algorithm proposed by Fagin et al[6]. This approach has four different cases.

The first case is when both i and j appear in τ and ω. In this case the method utilizes Equation 11 but only on the items in the set, Z.

The second case is when i and j both appear in τ or ω but only i or j appears in the other. The number of cases this apply can be calculated according to Equation 12. The equation sums the item positions from the sets S and T in the lists τ and ω and subtracts it from $|\tau \cup \omega| + 1$ which is multiplied by $|S|$.

$$(k-z)(k+z+1) - \sum_{i\in S}\tau(i) - \sum_{i\in T}\omega(i) \quad (12)$$

The third case is when i appears in one list and j in the other. The result of the third case is calculated by $(k-z)^2$, which is the length of the lists minus the intersection, to the power of 2.

The fourth case is when both i and j appear in one list but not the other. In this case Equation 13 is used. p in this case is a penalty value between 0 and 1. As the method we use is an average approach this value is 0.5. p is multiplied with the binomial coefficient of the length of different items in the top-k lists.

$$2p\binom{k-z}{2} \quad (13)$$

Combining these cases into one method, we get the K_{Haus} algorithm which can be seen in Equation 14.

$$K_{Haus}(\tau,\omega) = \frac{1}{2}(k-z)(5k-z+1) + \sum_{i,j\in Z} K_{i,j}(\tau,\omega) + \sum_{i\in S}\tau(i) - \sum_{i\in S}\tau(i) \quad (14)$$

The result of the K_{Haus} is normalized by dividing it by $n(n-n)/2$, which gives an approximation of the average distance between the lists. It is an approximation because in case four, the method assumes that there is an equally large chance of the items being in the correct order. Due to this, the method returns 0.78 if the lists are completely disjoint. If the lists are reverse of each other it scores 1 and 0 if the lists are equal.

Spearman's Footrule Distance

Another distance measure we use is the Spearman's Footrule Distance(SFD) [6]. SFD finds the exact distance between item i in two different ranked lists containing i. The way it finds this item distance is by subtracting the item indexes from each other as can be seen in Equation 15.

$$F(\tau,\omega) = \sum_{i=1}^{k} |\tau(i) - \omega(i)| \quad (15)$$

As we work with partial lists we use an alternate version called F_{Haus}, see Equation 16, proposed by Fagin et al[6]. As the lists τ and ω can contain different items, the missing index values for items are replaced by ℓ which is some value larger than k, as it follows that they would be outside the top-k list. Based on the article by Fargin et al we set ℓ to be equal to $(3 * k - z + 1)/2$.

$$F_{Haus}(\tau,\omega) = (k-z)(3k-z+1) + \sum_{i\in Z}|\tau(i)-\omega(i)| - \sum_{i\in S}\tau(i) - \sum_{i\in Z}\omega(i) \quad (16)$$

In order to normalize we divide the result of Equation 16 by $n^2/2$ which is the maximum value of the algorithm. Doing so we get a value of 0 if τ and ω are in the same order or 1 if the lists are the reverse of each other or completely disjoint.

4.1.6 T-test

We made paired t-tests for all methods[10]. Each method is compared with each of the other methods for all measures. The t-test outputs a p-value, which is the probability that the difference between two sets of results is coincidental. At 0.05 or lower, it is considered that there is statistically significant difference in the means of the two sets.

4.2 Results

nDCG

Figure 3 shows the nDCG score for BC, MC_4, SF, and Avg. For nDCG a higher score is better and is within 1 to 0. All methods see a sharp drop off in the quality of their recommendations as the group sizes increase. As shown in Table 2, BC drops the most in the jump from 4 to 8 group members, however it also have the best results, and outperforms all other methods across all group sizes. MC_4 is the second best and follows the same trend and quickly plateaus in score. One outlying case SF starts out close to Avg for a group size of four, but retains a higher score and is closer to BC and MC_4 as the size increases. Avg is the worst performing overall.

Another trend is observable in Table 2. The highest scoring method is also dropping the most in score, aside from Avg from group size 4 to 8. This effect is visible on all group sizes for all methods.

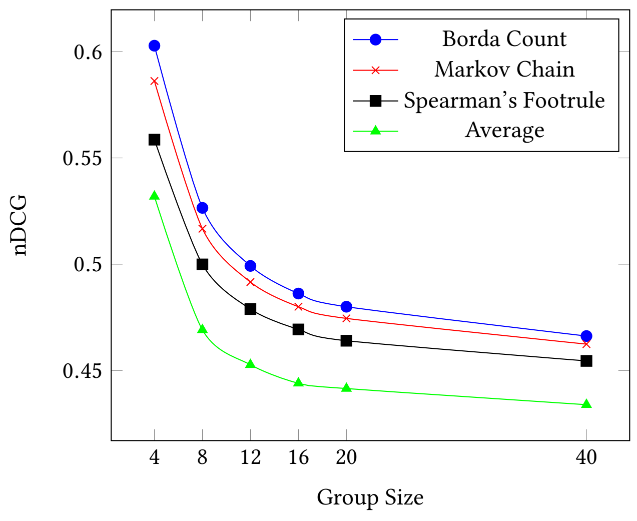


Figure 3. Results using nDCG

	4 to 8	8 to 12	12 to 16	16 to 20	20 to 40
BC	12.66	5.19	2.6	1.28	2.88
MC	11.86	4.86	2.36	1.15	2.55
SF	10.51	4.20	2.00	1.13	2.05
Avg	11.81	3.50	1.92	0.56	1.72

Table 2. Percentage decrease between the groups for nDCG

The p-values for the t-test for nDCG is shown in Table 3. Our paired t-tests show that all results for the nDCG measure are statistically different from each other.

Any zeros in the table are considered to be so small that it was rounded down and is some small non-zero value.

	4	8	12	16	20	40
BC/MC	$3e^{-270}$	$3e^{-234}$	$2e^{-220}$	$4e^{-218}$	$1e^{-213}$	$3e^{-205}$
BC/SF	0	$2e^{-296}$	$7e^{-272}$	$4e^{-249}$	$1e^{-237}$	$1e^{-227}$
BC/Avg	$2e^{-310}$	0	0	0	0	0
MC/SF	$2e^{-203}$	$2e^{-166}$	$3e^{-142}$	$2e^{-129}$	$1e^{-130}$	$2e^{-133}$
MC/Avg	$5e^{-228}$	$5e^{-273}$	$3e^{-278}$	$5e^{-289}$	$1e^{-309}$	0
SF/Avg	$2e^{-72}$	$2e^{-138}$	$2e^{-147}$	$6e^{-165}$	$9e^{-166}$	$5e^{-211}$

Table 3. P-values for the nDCG t-test

Rating nDCG

In Figure 4 we see the scores of the Rating nDCG measure. All methods generally have high levels of satisfaction according to the measure, with none scoring below 95 percent satisfaction.

Avg is ahead of the other methods, but it is also the only method using the average rating of all the candidate items for its recommendation. The remainder are mostly identical in performance to each other.

As can be seen in Table 4, BC, MC_4, and SF do not decrease in score from 16 to 20, and SF increases in score. In general, the fall in nDCG score is extremely low compared to the other measures. This could indicate, that the individual recommendations are biased towards some selection of items.

Among the remainder, while MC_4 does outperform both BC and SF, the difference is small.

Table 5 shows the t-test results for Rating nDCG. For BC and SF, there are two cases, which has been marked in bold, where the difference is not significant enough to not be considered random. It can also be seen that the p-values for all but Avg are many orders of magnitudes smaller than seen for other measures, due to the similarity in the results.

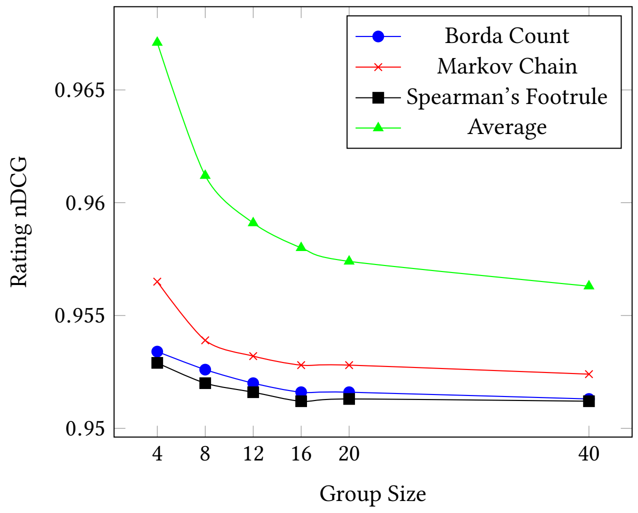


Figure 4. Results using Rating nDCG

	4 to 8	8 to 12	12 to 16	16 to 20	20 to 40
BC	0.084	0.063	0.42	0	0.032
MC	0.27	0.73	0.42	0	0.042
SF	0.094	0.042	0.042	-0.011	0.011
Avg	0.61	0.22	0.11	0.063	0.11

Table 4. Percentage decrease between the groups for Rating nDCG

	4	8	12	16	20	40
BC/MC	$5e^{-64}$	$5e^{-38}$	$2e^{-37}$	$3e^{-58}$	$3e^{-60}$	$5e^{-70}$
BC/SF	**0.058**	$1e^{-5}$	$1e^{-5}$	$2e^{-4}$	$5e^{-5}$	**0.089**
BC/Avg	$1e^{-251}$	$2e^{-275}$	$5e^{-299}$	0	0	0
MC/SF	$1e^{-49}$	$3e^{-52}$	$2e^{-55}$	$7e^{-62}$	$6e^{-66}$	$8e^{-68}$
MC/Avg	$9e^{-221}$	$7e^{-247}$	$1e^{-266}$	$3e^{-296}$	$2e^{-308}$	0
SF/Avg	$5e^{-212}$	$1e^{-262}$	$6e^{-280}$	$3e^{-287}$	$4e^{-305}$	0

Table 5. P-values for Rating nDCG t-test

Kendall Tau Distance

Looking at Figure 5 we can see the results of the KTD test. As covered in Section 4.1.5 the distance measures have a score between 0 and 1 where 0 correspond to equal lists, 1 is that the lists are reverse of each other, and 0.78 is that the lists are disjoint as we used an average approach.

Looking at the approaches individually Avg clearly scores the highest. The reason is that Avg disregards the item ranks in the top-k lists and aggregates them based on the average rating between the group members instead. SF performs worse than both MC_4 and BC, which could be seen as having to do with SF ranking its candidates using a median like approach, which is closer to how Avg performs. Lastly, performing best and almost equal we have BC and MC_4. Worth noting is that when the groups are small MC_4 performers slightly better than BC, but already at group size 8 BC out scales MC_4.

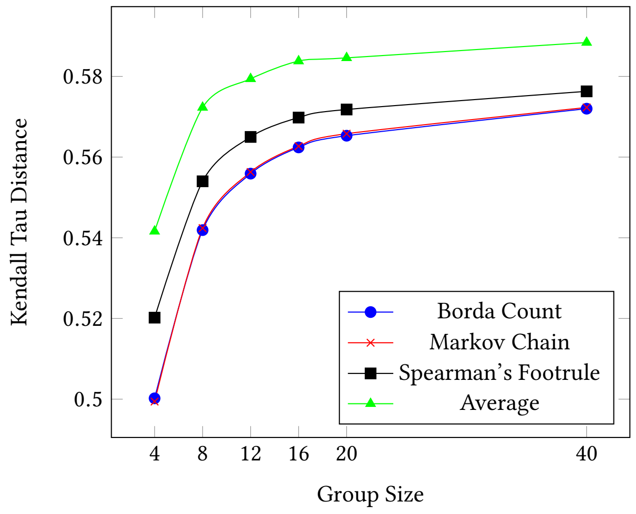


Figure 5. Results using KDT

When looking at the percentage distance increase between group sizes in Table 6, it can be seen that all the methods follow the same trend across the group sizes. There is a large difference with an average increase of 7.28 percent between group sizes 4 and 8, as the groups grow the increase in the KTD quickly fades and becomes very low between groups. By the time we reach groups 20 and 40 the average change in distance is only 0.945 percent.

For the t-tests shown in Table 7, all results are shown to be statistically significant in their differences, except for one case between BC and MC. The remainder are different enough to be statistically significant, but the p-values are still considerably higher than usual between BC and MC for a sample of this size.

	4 to 8	8 to 12	12 to 16	16 to 20	20 to 40
BC	8.34	2.58	1.17	0.52	1.19
MC	8.61	2.56	1.15	0.55	1.15
SF	6.50	1.99	0.85	0.35	0.79
Avg	5.67	1.24	0.76	0.14	0.65

Table 6. Percentage increase between the groups for KDT

	4	8	12	16	20	40
BC/MC	0.038	0.031	0.02	**0.071**	$9e^{-5}$	0.002
BC/SF	$2e^{-201}$	$1e^{-231}$	$1e^{-236}$	$7e^{-226}$	$5e^{-227}$	$1e^{-198}$
BC/Avg	$3e^{-276}$	$4e^{-296}$	$3e^{-305}$	$9e^{-303}$	0	0
MC/SF	$2e^{-230}$	$8e^{-239}$	$6e^{-239}$	$7e^{-239}$	$2e^{-226}$	$4e^{-216}$
MC/Avg	$4e^{-259}$	$2e^{-283}$	$7e^{-298}$	$7e^{-303}$	$4e^{-321}$	0
SF/Avg	$1e^{-93}$	$1e^{-143}$	$5e^{-159}$	$2e^{-177}$	$3e^{-199}$	$3e^{-261}$

Table 7. P-values for KTD t-test

Spearman's Footrule Distance

In Figure 6 are the results of the SFD tests. As in KTD, the distance is between 0 and 1, and 0 represents the perfect match.

SF performs the best in this test, which is unsurprising. SF has a natural advantage over the other methods when using SFD as SF works to minimize distance per the SFD principle. Avg again performs the worst when looking at the SFD results. This is for the same reason as with the KTD measure, Avg does not take the rank of items into account. For MC_4 and BC we get some interesting results. They still follow each other really close. However in this test MC_4 performs marginally better in the most cases except at group size 12 and 16 where BC is slightly better.

In general the tendencies are very similar to those of KTD and the approaches follow the same curve with all most the same distance jumps between the group sizes. This can be noted in Table 8. The best performing methods do fall faster, but it is relative to its performance.

Table 9 shows the t-tests for SFD. BC and MC have two cases where the results are not distinct enough to be statistically different. BC and MC for group size 12 show the highest likelihood of of all comparisons to be indistinguishable.

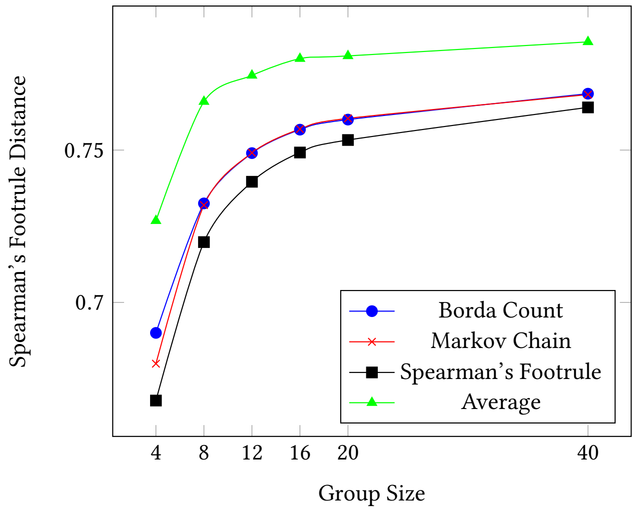


Figure 6. Results using SFD

	4 to 8	8 to 12	12 to 16	16 to 20	20 to 40
BC	6.16	2.25	1.03	0.44	1.12
MC	7.65	2.34	1.04	0.46	1.03
SF	7.79	2.75	1.30	0.55	1.42
Avg	5.38	1.12	0.71	0.12	0.59

Table 8. Percentage increase between the groups for SFD

	4	8	12	16	20	40
BC/MC	$5e^{-86}$	0.027	**0.875**	**0.222**	0.0145	0.001
BC/SF	$4e^{-265}$	$3e^{-288}$	$7e^{-295}$	$2e^{-274}$	$3e^{-277}$	$2e^{-275}$
BC/Avg	$7e^{-176}$	$2e^{-254}$	$5e^{-249}$	$4e^{-253}$	$5e^{-269}$	$5e^{-297}$
MC/SF	$7e^{-42}$	$2e^{-243}$	$1e^{-250}$	$6e^{-258}$	$2e^{-246}$	$1e^{-242}$
MC/Avg	$2e^{-217}$	$2e^{-250}$	$2e^{-248}$	$2e^{-249}$	$1e^{-265}$	$1e^{-305}$
SF/Avg	$2e^{-288}$	0	0	0	0	0

Table 9. P-values for SFD t-test

5 Discussion

Across all the measurements, MC_4 is almost identical in performance to that of BC. As the underlying heuristic of the MC_4 method is the Copeland Method, it raises the possibility that other extensions of Markov Chain can achieve even greater results.

However, for MC_4, other factors such as complexity and speed limits the utility of it compared to BC, as BC is both simple to implement and can be implemented in linear time whereas MC_4 is slower. However, optimizations for MC_4 and its variants exists which run in quadratic time which is a significant running time improvement[4].

We saw a trend of the nDCG score being a good indicator for how well the same methods performed for the distance measure. Though SF performed well in SFD, but it fell behind on other measures, with the same to be said for Avg and its performance for Rating nDCG. As such it is possible that nDCG is not a better measure than SFD or

KTD for measuring the best group recommender, and might simply favor BC and MC_4.

Of all the measures, Avg has the worst results overall aside from Rating nDCG. The results from Rating nDCG in isolation sees Avg perform the best. So if Rating nDCG is a better measurement because it does not consider the rankings, and instead looks directly at the ratings given by the users, then Avg is providing the better recommendations.

However, results from Rating nDCG for Avg and the other methods are so close that there is little practical value for one method to the other. Should it be the case that Rating nDCG is the best measure, we have a theory about rearranging the users top-k list and order them according to average. We will expand on this in future work in Section 6.1.

Baltrunas et al found that the more alike a group is, the more effective the group recommender[1]. Likewise for our setup, it is likely that the results can vary depending on the recommender used in the individual recommendations stage. It follows that biases introduced by the individual recommendation can result in an either more or less alike-thinking group for the same dataset.

6 Conclusion and Future Work

In this paper we have evaluated several aggregation methods for group recommendations. Our findings are simple to reproduce, and give a good indication of the performance of the various methods tested. The best performing aggregation method was BC per our measures and setup. The multiple measures we use also reinforce these results aside from Rating nDCG.

We worked with Markov Chains, specifically MC_4, which to the best of our knowledge have not been tested for the group recommendation domain, and it performed almost on par with BC.

We got results similar to that of Baltrunas et al for their setup, and found that the rate of decrease in quality does not continue at the same rate beyond a group size of 8, and that the rate of change decreases sharply and is small at group sizes 16 and above. We confirmed the same trend for all the measures tested.

6.1 Future Work

Measurements on Real Data

To address the issue of nDCG as a satisfaction measure for group recommendations there is a need for real data. When training and evaluating our recommender system, we had to make do with individual ratings and make assumptions about what makes for good measures. With data on how people make recommendations, we could make some more informed conclusions on the used measures for group recommendations. There are several ways one could go in acquiring data. The first is to test group recommender methods on people, and have them give feedback on the results of the recommendation. This provides an indication of how the method performs for the group by aggregating the individual scores. Another way is to have people make recommendations based on information given about a group and test how close various methods come to these. The latter assumes that humans make good group recommendation systems and are consistent about fairness or gravitate towards better aggregation methods, but it is also easier to generate large amounts of data on.

BC and MC_4 Extensions

BC and MC_4 exist in many variants, and repeating our experiments with other extensions can reveal more about the measures and the extensions. Candidates could be other extensions presented by Dwork et al[4] for Markov Chain, or others for Borda Count.

Context and Influence

Research on the effect of including influence and contextual information to improve recommendations for groups has been done by Quintarelli et al[13]. The idea is that certain persons have more influence in specific contexts. Quintarelli et al gives the example of a family consisting of young kids and their parents how watches television together. Depending on the time of day, the influence change between the parents and kids, as the kids maybe have a higher influence in the afternoon when there are many kid friendly programs available, but in the evening the parents have the most influence in order to censor for inappropriate programs for minors.

This idea of context and influence could help give more appropriate recommendations which could be a great way to improve on recommendations.

Reordering of Ranked Lists

A pre-ranked aggregation method we did not test in our report was the reordering of the ranked lists. The idea is to rearrange the rankings by the average rating from the other users before performing the aggregation step to better account for the opinions of other users. It is our hypothesis that it would lead to better group satisfaction overall in the case that Rating nDCG is a good measure.

7 Acknowledgements

We extend our gratitude to our supervisor, Peter Dolog, for his exemplary help during this project. We would like to thank Claus Nygaard Madsen, whom we have collaborated with on the foundation of this project.

Lastly, we are grateful of Aalborg University for its support, and for funding our survey aspirations.

References

[1] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. 2010. Group Recommendations with Rank Aggregation and Collaborative Filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 119–126. https://doi.org/10.1145/1864708.1864733

[2] Shlomo Berkovsky and Jill Freyne. 2010. Group-based Recipe Recommendations: Analysis of Data Aggregation Strategies. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 111–118. https://doi.org/10.1145/1864708.1864732

[3] Lukas N. D. Claus N. M., Lasse D. C. 2016. *Group Recommendation Using Voting as Mediator*. Technical Report. Department of Computer Science Aalborg University, Aalborg, Denmark.

[4] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank Aggregation Methods for the Web. In *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*. ACM, New York, NY, USA, 613–622. https://doi.org/10.1145/371920.372165

[5] Ward Edwards and Barbara Fasolo. 2001. Decision technology. *Annual review of psychology* 52, 1 (2001), 581–606.

[6] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing Top K Lists. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '03)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 28–36. http://dl.acm.org/citation.cfm?id=644108.644113

[7] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A Free Recommender System Library. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. ACM, New York, NY, USA, 305–308. https://doi.org/10.1145/2043932.2043989

[8] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

[9] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446. https://doi.org/10.1145/582415.582418

[10] Samuel Kotz and Norman L. Johnson (Eds.). 1992. *The Probable Error of a Mean.* Springer New York, New York, NY, 33–57. https://doi.org/10.1007/978-1-4612-4380-9_4

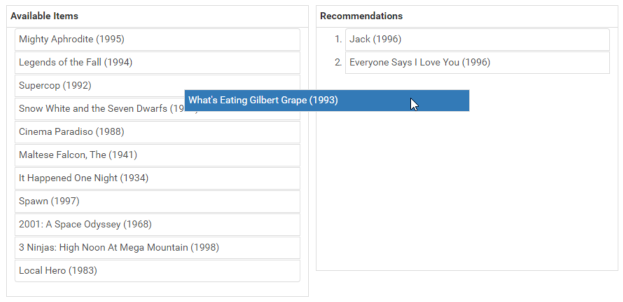
[11] Judith Masthoff. 2004. Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *User Modeling and User-Adapted Interaction* 14, 1 (2004), 37–85. https://doi.org/10.1023/B:USER.0000010138.79319.fd

[12] J. Munkres. 1957. Algorithms for the Assignment and Transportation Problems. *Journal of the Society of Industrial and Applied Mathematics* 5, 1 (March 1957), 32–38.

[13] Elisa Quintarelli, Emanuele Rabosio, and Letizia Tanca. 2016. Recommending New Items to Ephemeral Groups Using Contextual User Influence. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 285–292. https://doi.org/10.1145/2959100.2959137

[14] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2015. *Recommender Systems Handbook.* Springer-Verlag New York, Inc., 752–757.

[15] Donald G. Saari and Vincent R. Merlin. 1996. The Copeland method. *Economic Theory* 8, 1 (1996), 51–76. https://doi.org/10.1007/BF01212012



Available Items

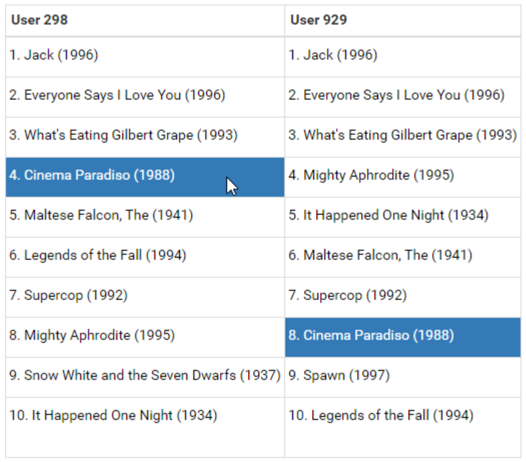
- Mighty Aphrodite (1995)
- Legends of the Fall (1994)
- Supercop (1992)
- Snow White and the Seven Dwarfs (19
- Cinema Paradiso (1988)
- Maltese Falcon, The (1941)
- It Happened One Night (1934)
- Spawn (1997)
- 2001: A Space Odyssey (1968)
- 3 Ninjas: High Noon At Mega Mountain (1998)
- Local Hero (1983)

What's Eating Gilbert Grape (1993)

Recommendations

1. Jack (1996)
2. Everyone Says I Love You (1996)

Figure 7. The list of available items and the box holding the recommendations of the survey participant



User 298	User 929
1. Jack (1996)	1. Jack (1996)
2. Everyone Says I Love You (1996)	2. Everyone Says I Love You (1996)
3. What's Eating Gilbert Grape (1993)	3. What's Eating Gilbert Grape (1993)
4. Cinema Paradiso (1988)	4. Mighty Aphrodite (1995)
5. Maltese Falcon, The (1941)	5. It Happened One Night (1934)
6. Legends of the Fall (1994)	6. Maltese Falcon, The (1941)
7. Supercop (1992)	7. Supercop (1992)
8. Mighty Aphrodite (1995)	8. Cinema Paradiso (1988)
9. Snow White and the Seven Dwarfs (1937)	9. Spawn (1997)
10. It Happened One Night (1934)	10. Legends of the Fall (1994)

Figure 8. Two lists of movie preferences for a user

Appendix A Survey

In order to make a dataset in a short time frame, we looked into crowdsourcing, where it was possible to pay people to answer surveys or do tasks that computers cannot. In particular we were looking at Clickworker and Mechanical Turk, which allowed external surveys. Since no sites had an inbuilt survey creation tool that could handle our requirements, we decided to make our own survey website.

Using this method, we could potentially reach thousands of people without limiting us to the survey tools made available on the standard websites.

For web hosting, we went to DigitalOcean, which offered a Ubuntu server setup with Django. We added Gunicorn as the WSG interface and nginx as the http server. Behind it all, we had a MySql database keeping track of the survey questions and results along with timestamps.

The survey itself asks participants to personally give a recommendation to a group of users. The participant knows each user's own top 10 preference, and must decide on their own what aggregation strategy they wish to follow. An important aspect of the survey was making it as easy to complete as possible. As we had to pay each participant, each such improvement could be translated to a saving, which in turn translated to a larger and more useful dataset. So to make it more intuitive and not overload each user with information, we made it so that hovering the mouse over a movie title will make the movie's position light up on all the other users' rankings, shown in Figure 8, and gave the user a tooltip about other movie positions. Additionally, we made the ranking system into a drag-and-drop, such that the participant could drag and easily rearrange their recommendations. This can be seen in Figure 7.

After making a recommendation, the user could proceed to the next step, and upon completing all steps they would reach a screen providing them with a code. With each step, the group size is increased by one, going from 4 to 8 users. The code is important, as the participant must present this as evidence to the crowdsourcing site as proof of their participation. For our survey, we decided to generate a unique code for each participant mixing the assigned groups and a timestamp, so that we could deduce which user responded when and with what. This precaution was necessary so that it was possible to filter out participants rushing through the survey with no care for their answers.

Also, since we wanted to have a balanced dataset, we separated our groups into 40 sets of groups of size 4 to 8, and made it so that every user would get a randomly picked set. The database would keep track of how many responses each set had, such that we could prioritize the sets with fewer responses and get a balanced dataset. It would also mean that anyone taking the test twice would be unlikely to see the same survey.

Before running the survey on Mechanical Turk, we ran it past some other willing participants for evaluation and decided to halve the number of groups each participants would give recommendations to from 10 to 5, due to feedback about the length of the survey.

For the crowdsourcing website, we ended up going with Amazon's Mechanical Turk, as it is the more well-known and cheaper service. When ready, we injected a good amount of money on the account, as one had to prepay for any work requested and started up a limited run to test out the services and find a suitable price range. We managed to get a few responses. On Mechanical Turk, the participant would see our survey, click in and be provided a link to our survey. Upon completion of the survey, our participant would get the code and input it on the Mechanical Turk website. Initial results were interesting with big differences between how much time participants spent on the survey. We noted a few obvious cheaters who blazed through a survey in seconds thanks to the timestamps, however as a requester and with the timestamps in our database, we would be able to sort them out, and we could also reject their work on the Mechanical Turk site.

Though, on the third day of this limited run we ran into problems with Amazon Mechanical Turk. We were unable to access our account, and had to contact their support team. Soon enough the support team responded that our account had been suspended, and that we would be informed about the reason for the suspension in 2-3 days along with fate of our account. In two days time, Mechanical Turk support wrote us that our account had been suspended indefinitely and our current survey stopped because of a violation of the participation agreement. Unaware of any possible violations we could have made, we made further inquires, but never got a clear answer. Additionally, our funds on the account were also confiscated. So in the end, we ended up only getting responses totaling a few dollars worth of data.

Appendix B Borda Count Extensions

In this section we will comment on some extensions for BC called Borda Weighted Count(BWC), Borda Transferable Count(BTC), and Borda Escalating Count(BEC) which were made during last semesters work and show promising results at that time[3].

Unfortunately an unwanted feature was present in during the original nDCG test in the earlier work and after seeing the results of the preliminary tests for this paper we decide to drop the extensions for now.

Looking at the results in the Figures 9, 10, 11, and 12 it is clear to see that BC on most cases performs best or equal to the other approaches though worth noting is that BWC performs almost equal to BC in most cases and even slightly better in a few cases. These results shows that there still could be made possible improvements on BC.

For a further explanation of the BC extensions we reference to our technical report[3].

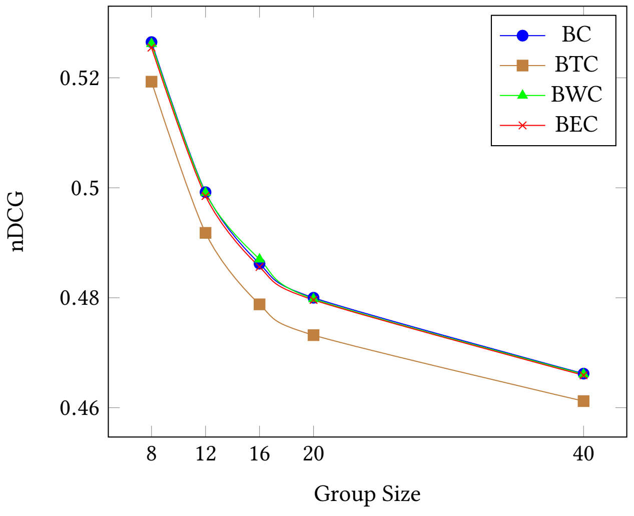


Figure 9. Results using nDCG on BC extensions

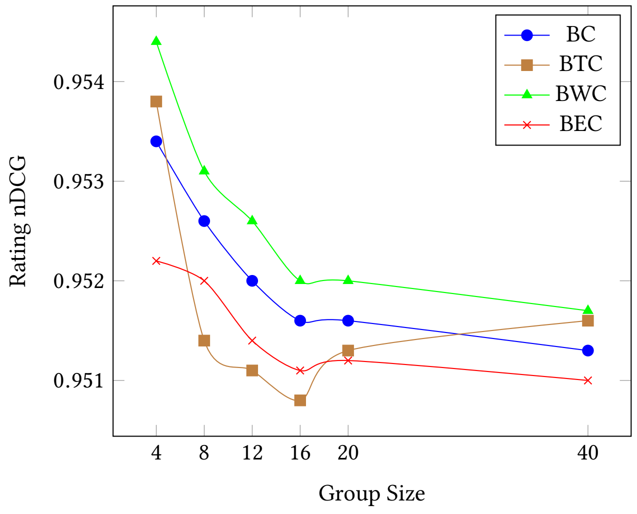


Figure 10. Results using Rating nDCG on BC extensions

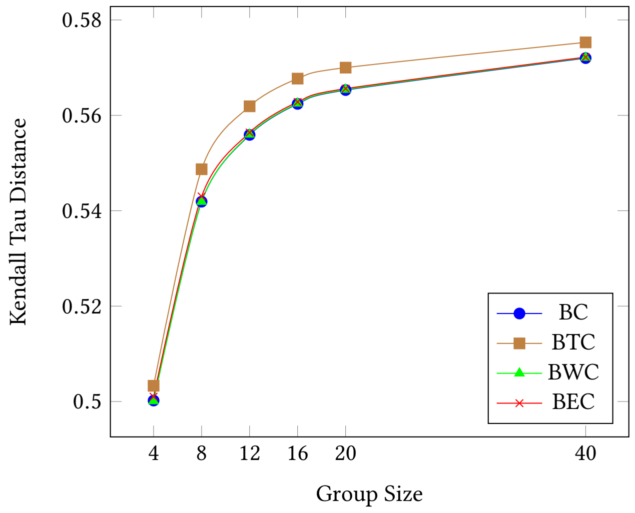


Figure 11. Results using KTD on BC extensions

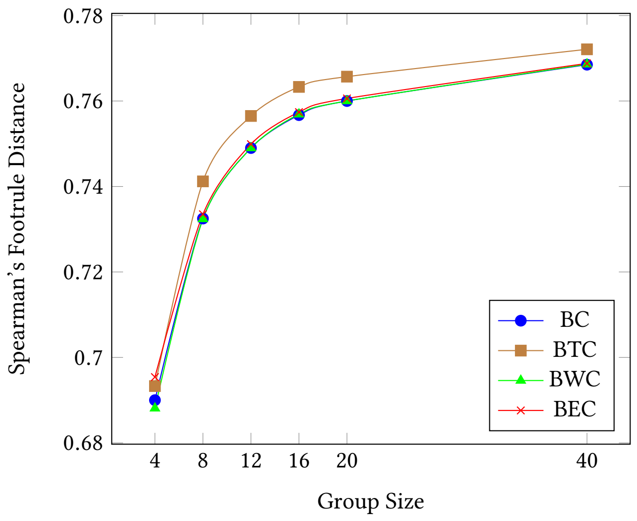


Figure 12. Results using SFD on BC extensions