INVESTIGATING COMPARISON-BASED EVALUATION FOR SPARSE DATA

by

Jose Antonio Martinez Torres, B.E.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2014

Committee Members:

Byron J. Gao, Chair

Anne H.H. Ngu

Yijuan Lu

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

ACKNOWLEDGEMENTS

I would like to thank Dr. Byron Gao not only for involving but motivating me to participate in his research, as well for his enriching advice. I also appreciate Dr. Anne H.H. Ngu and Dr. Yijuan Lu for being involved in the thesis committee during the process. Finally, I would like to thank my family for their unconditional support throughout my education.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

Evaluation is ubiquitous. Often we need to evaluate a set of target entities (movies, restaurants, products, courses, paper submissions) and obtain their true ratings (average ratings from the population) or true rankings (rankings based on true ratings). Based on the law of large numbers, average ratings from large samples can well serve the purpose. However, in practice evaluation data are typically extremely sparse and each entity would receive a very small number of ratings from evaluators. In this case, the average ratings would significantly differ from the true ratings due to biased distributions of evaluators holding different standards or preferences. Based on the observation that comparative evaluations (e.g., paper 1 is better than paper 2) are more trustworthy than isolated ratings (e.g., paper 1 has a score of 4.5), in this study we investigate comparison-based evaluation, where the principle idea is to first extract a partial ranking for the entities evaluated by each evaluator, and then aggregate all the partial rankings to obtain a total ranking that well approximates the true ranking. The aggregated total ranking can be used to further estimate the true ratings. In this study we also investigate an associated topic of evaluation assignment (assigning target entities to evaluators). In many applications (e.g., academic conferences) there is such an assignment phase before evaluation is conducted. Currently in these applications assignment is not sophistically designed to maximize evaluation quality. We propose a layered assignment approach to maximize the quality of comparison-based evaluation for given evaluation resources

(evaluation is generally labor-intensive). All the proposed algorithms have been implemented and validated using benchmark datasets in comparison with state-of-the-art methods. In addition, to demonstrate the utility of our approach, a prototype system has been deployed and made available for convenient public access.

1. INTRODUCTION

Evaluation is ubiquitous, where we need to evaluate a set of interested entities (movies, restaurants, products, courses, paper submissions) and obtain their true ratings or true rankings. We refer to these entities as evaluatees or target entities. We define true ratings as average ratings from the population. Statistically, a population is a complete set of items that share at least one property in common that is the subject of a statistical analysis. In our case, a population can be considered as the entire group of people who interact with the interested target entities and from whom we can draw evaluation conclusions about the entities. For example, a population with respect to a movie can be considered as all the people who have an interest in (have or have not watched) the movie. A true ranking is a total ranking for the set of target entities based on their true ratings. In practice, obtaining the true ranking is often more important than obtaining the true ratings in making decisions/selections.

Common evaluation procedure: Since it is virtually impossible to collect evaluations from the entire population, true ratings and true rankings can never be obtained. They can only be estimated/approximated. As a common evaluation procedure, a sample of the population will be chosen as evaluators or reviewers to evaluate/review the target entities. Typically, an evaluation/review can be represented as a rating score within a certain range (e.g., 1 to 5). Then the average ratings from the sample will be calculated and used as evaluation scores for the target entities.

As known from the law of large numbers (Grimmett & Stirzaker, 1982), this procedure works fine if the sample size is large, where each entity receives a large number of ratings. In this case, the sample would well reflect the distributions of

standards/preferences/opinions/evaluations of the population and average ratings from the sample would closely approximate the average ratings from the population, e.g., the true ratings, serving as an unbiased indicator of quality (Marsh & Roche, 1997). In the following, we use average ratings to particularly refer to the average ratings from the samples unless otherwise specified.

Sparse data challenge: A practical and critical challenge for the common evaluation procedure in reality is that evaluation data are typically extremely sparse, where each target entity only receives very few ratings (although there can be a decent size of total number of evaluators), and each evaluator only evaluates very few target entities (due to capacity constraints). Evaluation is labor-intensive and we cannot afford to have each evaluator to evaluate all the target entities. For example, suppose a computer science conference has 500 paper submissions to be reviewed by 150 selected professional reviewers. Each reviewer cannot review all the submissions. Typically they will be assigned several papers to review. Suppose each reviewer is assigned 10 papers, then each paper would receive on average 3 reviews from different reviewers.

In this case, since the evaluators/reviewers typically hold different standards/criteria/preferences in the evaluation/review process, average ratings would significantly differ from true ratings due to biased distributions of standards/preferences from the evaluators. This is what we call the evaluation bias problem, which is illustrated in Figure 1. The figure shows the relationship between error rate and sample size, where error rate is based on the Kendall tau distance metric and computed between the average rating-based ranking and the "true ranking" (average rating from the largest sample). From the figure we can see that the error rate increases as the sample size decreases.

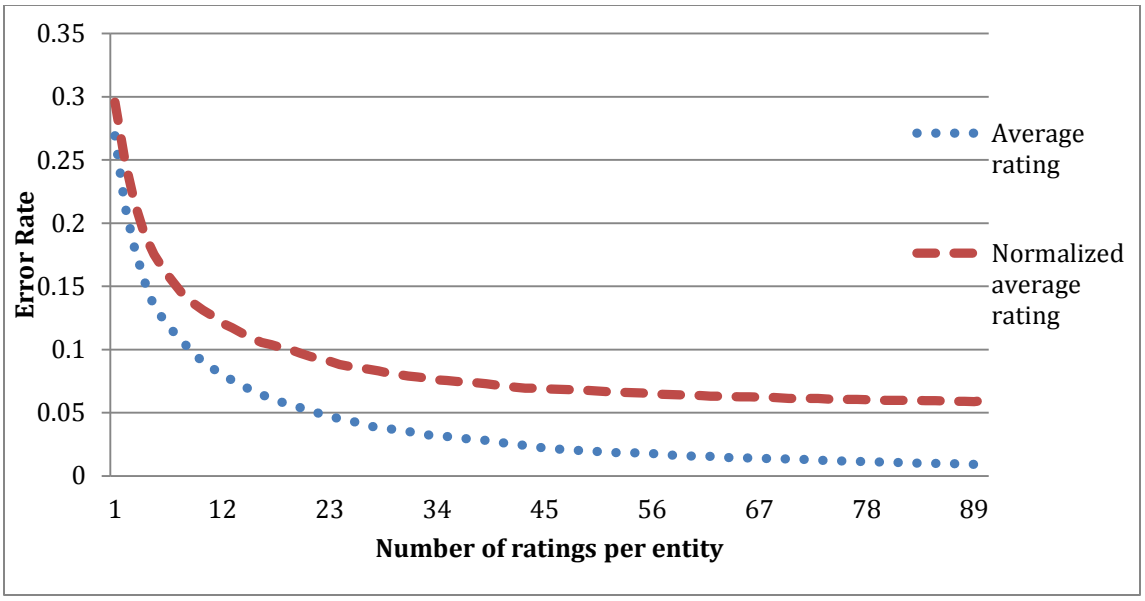

Figure 1: Error rate vs. sample size

One possible straightforward solution to remove the evaluation bias is to standardize/normalize evaluations for the evaluators. In particular, we can normalize all the rating scores given by an evaluator, which would have the effect of forcing a common standard for all the evaluators. However for this approach to work, each evaluator has to evaluate a large number of target entities that well represent the set of all target entities, which is impractical and would not happen in our sparse data setting. From Figure 1, we can observe that the error rate increases as the sample size decreases.

Comparison-based evaluation: For average ratings to be correct and fair, the following assumption has to hold: All the evaluators hold the same standard/preference at all times. In practice, this assumption is not reasonable. Obviously, different evaluators would significantly differ in their standards/preferences/criteria for various reasons in the evaluation process. However, it is reasonable to assume that each evaluator holds the same standard/preference at all times. For example, we can reasonably assume that each paper reviewer holds the same standard in reviewing the three papers assigned to her.

Therefore, the relative and comparative evaluation from an evaluator is more trustworthy than the absolute ratings given by her. Based on this observation, we conduct the REVA project in this thesis work that investigates comparison-based evaluation.

In particular, in the REVA project we design comparison-based evaluation algorithms that can effectively and efficiently eliminate evaluation bias and approximate true rankings and true ratings for sparse evaluation data. The principle idea of the REVA approach is to first extract a partial ranking for the entities evaluated by each evaluator, and then aggregate all the partial rankings to obtain a total ranking that well approximates the true ranking. A ranking is partial if it does not cover all the target entities, e.g., a ranking of 3 papers out of 500 assigned to a particular reviewer. A ranking is total if it covers all the target entities. We formulate rank aggregation as an optimization problem that computes an optimized total ranking minimizing a given rank distance metric such as Kendall tau distance or weighted Kendall tau distance. Such distance metrics are used to measure the distance between two ranked lists. Since the exact rank aggregation problem is NP-hard, we focus on designing efficient heuristic algorithms in REVA. Once a quality aggregated total ranking is obtained, it can be used to further estimate the true ratings.

Evaluation assignment: In the REVA project we also investigate a closely related topic of evaluation assignment. In many applications (e.g., academic conferences) before evaluation is conducted there is an assignment phase that assigns target entities to evaluators. To our best knowledge, currently in none of these applications assignment is sophistically designed to maximize evaluation quality. Evaluation is generally labor-intensive. Without intelligent assignment, a lot of evaluation effort would be wasted on obvious/easy comparisons (e.g., to compare two items with a big difference in quality).

On the other hand, insufficient evaluation effort would be given to subtle/hard comparisons (e.g., to compare two items with a little difference in quality).

In REVA we propose a layered assignment approach to maximize the quality of comparison-based evaluation for given evaluation resources, leading to improved approximation to ground truth. The main idea is to perform evaluation round by round (thus layered or multi-phase). In each round, assign several comparable entities to each available evaluator, where entities are comparable is they receive similar evaluations from the previous round. The assignment of comparable entities to the evaluators leads to maximized total ranking improvement from round to round. On the other hand, the improved total ranking also leads to better assignment from round to round because the comparability of the comparable entities will be more and more justified. As far as evaluation quality is concerned, it is desirable to use more layers/evaluation phases. However, this would prolong the whole evaluation process. In practice, the number of layers can be predetermined based on the given time constraint. Note that our layered assignment approach generalizes the existing conventional single layer assignment approach.

Implementation, validation, and demonstration: All the proposed algorithms as well as the comparison partners have been implemented. The algorithms have been validated using benchmark datasets in comparison with state-of-the-art methods. In addition, to demonstrate the utility of our approach, a prototype system has been deployed and made available for convenient public access through http://dmlab.cs.txstate.edu/reva/.

Contributions: In this thesis work, we make the following contributions:

- We investigate comparison-based evaluation for sparse evaluation data, leveraging trustworthy comparative evaluations to eliminate evaluation bias, leading to improved approximation towards true rankings and true ratings comparing to conventional evaluation methods.
- We also investigate a closely related topic of evaluation assignment, proposing a novel layered assignment approach that maximizes evaluation quality for given evaluation resources.
- We implement the proposed algorithms and validate their effectiveness and efficiency using benchmark datasets in comparison with state-of-the-art methods. To further illustrate the utility of our approach, we also deploy and maintain a demonstrative prototype that is available for convenient public access.

Outline: In Chapter 2, we survey the related work literature. In Chapter 3, we describe the REVA comparison based evaluation framework and algorithms. In Chapter 4, we describe the REVA layered evaluation assignment approach. In Chapter 5, we describe the implementation and the demonstrative prototype. In Chapter 6, we report experimental validation for REVA evaluation and REVA evaluation assignment. In Chapter 7, we conclude the thesis with a discussion of future work.

2. RELATED WORK

2.1 Rating Score Aggregation

Rating score aggregation aggregates rating scores from many individual evaluators into an overall quality measurement for target entities. The most straightforward approach is to take the average of rating scores. A more sophisticated approach is to consider reputation (e.g., evaluation capacity) of reviewers, and take the weighted average of rating scores, where more reputable reviewers are given higher weights. In (Riggs & Wilensky, 2001), the reputation of a reviewer is computed based on consensus (how closely the reviewer's rating scores are to the average scores). In (Chen & Singh, 2001), the reputation of a reviewer is computed based on opinions of other reviewers. However, reputable reviewers do not necessarily hold the same standard/preference.

Rating score normalization is a possible solution to overcome evaluation bias and achieve improved score aggregation. It converts rating scores of reviewers to a normalized scale. Most score normalization approaches assume that rating scores by each reviewer fit into a particular distribution (e.g., normal distribution) comparable to that of true rating scores. (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994) use z-score normalization (Walpole, Myers, Myers, & Ye, 2002), aiming to calibrate rating scores to a more equitable standard (Arkes, 2003). (Sarwar, Karypis, Konstan, & Riedl, 2000) perform normalization by subtracting the rating scores from a reviewer by the reviewer's average score. The subtraction can also be followed by LP normalization (Lemire, 2005) or by factor analysis (Traupman & Wilensky, 2006). However, in reality, the above

assumption does not hold for sparse evaluation data where each reviewer only reviews very few target entities.

Probability-based normalization approaches (Ferndandez, Vallet, & Castells, 2006), (Jin & Si, 2004) , (Jin, Si, Zhai, & Callan, 2003) convert rating scores to probability values, with the objective of removing the impact of outlier ratings. However, accurate probability estimation also requires that each reviewer is associated with many target entities, which is infeasible in our sparse data setting. Another line of related works (Lauw, Lim, & Wang, 2006), (Lauw, Lim, & Wang, 2007), (Lauw, Lim, & Wang, 2012) studied online collaborative rating. These works model leniency, a micro-behavior of individual reviewers, in order to achieve improved rating score aggregation. Since the leniency of a reviewer depends on scores of her own as well as co-reviewers, it is extracted by a mutual reinforcement model.

2.2 Ranking

Ranking a given set of target entities is an important task with vast number of applications in classification (Fürnkranz, Hüllermeier, Loza Mencía, & Brinker, 2008), (Saranli & Demirekler, 2001), clustering (Breitenbach & Grudic, 2005), (jun Zeng, cai He, Chen, ying Ma, & Ma, 2004), (Sun, et al., 2009), (Sun, Yu, & Han, 2009), web search (Page, Brin, Motwani, & Winograd, 1998), (Kleinberg, 1999), (Robertson, Walker, Beaulieu, Gatford, & Payne, 1995) databases (Chaudhuri & Das, 2003), (Chaudhuri, Das, Hristidis, & Weikum, 2004), (Mamoulis, Yiu, Cheng, & Cheung, 2007), (Wang & Su, 2002), software engineering (Inoue, et al., 2003), (Inoue, Yokomori, Yamamoto, Matsushita, & Kusumoto, 2005), (Gui & Scott, 2007), biology (Furlanello,

Serafini, Merler, & Jurman, 2010), (Chen, et al., 2010), (Liu, Yang, & Xu, 2010), and natural language parsing (Ng, 2005), (Collins, 2005), (Ng, 2005), just to name a few.

A ranking can be total or partial (Dwork, Kumar, Naor, & Sivakumar, 2001). A total ranking is a permutation of the given set of target entities. A partial ranking is a permutation of a subset of the set of target entities. A top k ranking is a special case of partial ranking that ranks the best k entities and ignore others. There are two well-known metrics that evaluate the distance between two permutations: the Kemeny optimality (Kendall & Gibbons, 1990) and the Spearman's foot rule distance (Diaconis, 1988). Other than these two classical metrics, (Kumar & Vassilvitskii, 2010) introduced some variants and a general definition considering specific conditions such as element weights, position weights and element similarities. (Fagin, Kumar, & Sivakumar, 2003) introduced some variants for the top k ranking problem.

2.3 Rank Aggregation

Rank aggregation is to combine ranking results of entities from multiple ranking functions in order to generate a better one (Liu, Liu, Qin, Ma, & Li, 2007). Rank aggregation has been extensively studied. We discuss several example approaches in the following. Borda Count (Borda, 1781) assigns a score corresponding to a rank or position within a ranking from an evaluator/voter, and then total or average scores can be used for evaluation purposes. Some recent works in the same line include Borda Fuse (Aslam & Montague, 2001) and median rank aggregation (Fagin, Kumar, & Sivakumar, 2003).

Some other works strive to find a total ranking while optimizing some ranking-based evaluation criteria, such as Kemeny optimality and Spearman's foot rule. Markov chain-based rank aggregation (Dwork, Kumar, Naor, & Sivakumar, 2001) introduced the

notion of a locally Kemeny optimal aggregation, a relaxation of Kemeny optimality, that ensures satisfaction of the extended Condorcet principle and yet remains computationally tractable.

Machine learning approaches have also been used for rank aggregation. (Liu, Liu, Qin, Ma, & Li, 2007) proposed a supervised learning method based on Markov chain for meta-search. (Klementiev, Roth, & Small, 2008) introduced an EM-based algorithm for learning parameters of the extended Mallows model (Mallows, 1957) without supervision.

2.4 Reviewer Assignment

The evaluation assignment problem that we study is related to the conference reviewer assignment problem. Nowadays academic conferences usually have an excessively high numbers of paper submissions. It is important to assign these papers to appropriate reviewers in the program committee so that the content of papers well matches the expertise and interests of reviewers. The primary input to this problem is a papers x reviewers matrix of "bids", expressing positive or negative interests of reviewers towards papers. The goal is to construct an optimized assignment taking into account reviewer capacity constraints, adequate numbers of reviews for papers, expertise modeling, conflicts of interest, and other global conference criteria (Conry, Koren, & Ramakrishnan, 2009).

There have been new interests in the Artificial Intelligence community for this problem since it was proposed as a "challenge" task at IJCAI-97. (Wang, Chen, & Miao, 2008) and (Goldsmith & Sloan, 2007) surveyed the recent studies. (Mimno & Mccallum, 2007) evaluated several methods for measuring the affinity of a reviewer to a paper.

(Dumais & Nielsen, 1992) studied reviewer assignment for the Hypertext conference. The paper observed that automated methods, including a variety of automated methods based on information retrieval principles and Latent Semantic Indexing, performed reasonably well in assigning relevant papers to reviewers but were still not satisfactory. The paper then proposed a new automated assignment method that achieved better performance than human experts by sending reviewers more papers than they actually have to review and then allowing them to choose part of their review load themselves. (Geller, 1997) described a program that automatically selects reviewers for submitted Artificial Intelligence papers. The program relies on a central knowledge/database of AI researchers and their areas of specialization. Several filter programs derive this knowledge from publication data and home pages on the World-Wide Web and from a well-known AI Genealogy. The Core Reviewer Selection Program relies on author-supplied subject areas to find reviewers. (Hettich & Pazzani, 2006) discussed a prototype application deployed at the National Science Foundation for assisting program directors in identifying reviewers for proposals. The prototype extracts information from the full text of proposals both to learn about the topics of proposals and the expertise of reviewers.

The reviewer assignment problem studied in these works focuses on matching the content of papers with the expertise and interests of reviewers. Our study of evaluation assignment is in an orthogonal direction, where we assume all papers equally match all reviewers, and our focus is on finding an assignment scheme that maximizes the quality of comparison-based evaluation for given evaluation resources, leading to improved approximation to true rankings and true ratings.

2.5 Recommender Systems

The evaluation problem we study is in the objective rating space (Traupman & Wilensky, 2006), where rating scores are primarily used to interpret the inherent quality of target entities. In contrast, recommender and collaborative-filtering systems (Lemire, 2005), (Shen, Lin, Xue, Zhu, & Yao, 2006) are in the subjective rating space, where rating scores are primarily used to interpret the preferences of reviewers.

Recommender systems generate item recommendations for a target user from a large collection based on the similarity between the target user and other neighboring users (Adomavicius & Tuzhilin, 2005), (Herlocker, Konstan, & Riedl, 2000), (Herlocker, Konstan, & Riedl, 2002). In recent years, recommender systems have become a must-own tool for e-commerce, such as eBay, Amazon, Last.fm, Netflix, Facebook, and LinkedIn. The two main paradigms for recommender systems are content-based filtering and collaborative filtering (CF). Content-based filtering makes recommendations by finding regularities in the textual content information of users and items, such as user profiles and product descriptions (Belkin & Croft, 1992). CF is based on the assumption that if users X and Y rate n items similarly or have similar behaviors, they will rate or act on other items similarly (Su & Khoshgoftaar, 2009). CF only utilizes the user-item rating matrix to make predictions and recommendations, avoiding the need of collecting extensive information about items and users. In addition, CF can be easily adopted in different recommender systems without requiring any domain knowledge (Liu & Yang, 2008). Ranking-based CF methods recommend items for users based on their rankings of items. In particular, such methods utilize similarity measures between two users based on their rankings on the same set of items. A common similarity measure is the Kendall tau

rank correlation coefficient (Marden, 1995). Recent efforts on ranking-based CF (Liu & Yang, 2008), (Liu, Zhao, & Yang, 2009), (Shi, Larson, & Hanjalic, 2010), (Weimer, Karatzoglou, Le, & Smola, 2007), (Kahng, Lee, & goo Lee, 2011) have demonstrated their advantages in recommendation accuracy.

The input data for content-based recommender systems contain profiles of items and users, which are not given in our evaluation problem. However, CF has a rating matrix as input, which is similar to our case. Our evaluation problem and CF differ in their objectives. While CF aims at computing item recommendations for the users, our evaluation problem aims at computing approximations of true ratings for the items.

Some recommender systems leverage pre-processing to correct certain global effects such as the number of ratings or the average ratings (Bell & Koren, 2007), or to perform data imputation replacing missing rating scores (Shen, Lin, Xue, Zhu, & Yao, 2006). Our problem targets elimination of evaluation bias and approximation of true ratings, which can potentially be used as an effective pre-processing step in recommender systems to improve recommendation quality.

3. REVA EVALUATION

REVA evaluation framework includes a robust heuristic algorithm that computes a better ranking that well approximates the true ranking. In this chapter, we will discuss REVA evaluation algorithm by including its main concepts and the variety of phases it can go through.

3.1 REVA Evaluation Overview

During the evaluation process, different evaluators will differ in standards, preferences and criteria for many reasons. We assume that the relative and comparative evaluation from each evaluator is more reliable than the absolute rating given by the evaluator. Based on this observation, we conduct the REVA project in this thesis work that studies comparison-based evaluation.

In REVA evaluation, we design comparison-based evaluation algorithm that is efficient and effective by removing evaluation bias and approximate the true ranking and true ratings for sparse data. The main idea of REVA evaluation approach is to first extract a partial ranking for the entities evaluated by each evaluator, and then aggregate all the partial rankings to obtain a total ranking that well approximate the true ranking.

We developed an efficient heuristic algorithm in REVA evaluation that generates good quality aggregated total ranking that can be used to further estimate the true rating.

REVA evaluation consists of different phases that compute the final ranking list. First, pairwise comparison amongst all entities is conducted using all evaluators' scores. After all comparisons have been completed, we then create a set of preferred directed pair that contains the pairs that are preferred by the evaluators based on the pairwise comparisons. Later, we calculate the *adjusted normalized difference* for each pair in the

set of preferred directed pairs and sort them in descending order. After we have completed this, we process all the preferred pairs into a set of compatible pairs by maintaining the transitive property, which allows us to make inferences, as well as ignore violations (e.g., trying to add the pair (C, A) but having the pair (A, C) already in the set). Based on this compatible pairs, a final ranking list can be generated. Finally, we conduct a benchmark between REVA evaluation and average rating against the true ranking.

3.2 REVA Evaluation Methodology

In this section, we will discuss in detail the main principles of REVA evaluation. The main phases of REVA evaluation are pairwise comparison, preferred directed pair set and compatible pair set generation, entities isolation, threshold setting and final ranking.

3.2.1 Pairwise Comparison

Pairwise comparison is the process of comparing a pair of entities in order to evaluate which is preferred. Pairwise comparison is commonly used in elections. The basic idea of pairwise comparison is that for each possible pair of candidates, one pairwise count specifies how many voters prefer one of the paired candidates over the other candidate, and other pairwise count specifies how many voters have the opposite inclination. This pairwise comparison method was designed to meet the fairness criterion used by the Condorcet method. For *N* entities, this will require $\frac{1}{2} N (N\text{-}1)$ pairwise comparisons. For instance, in every ten entities there will be forty-five pairwise comparisons that will be conducted.

Table 1: Partial matrix

	r_1	r_2	r_3	r_4
e_1	3	-	3	4
e_2	4	2	-	5
e_3	-	4	2	1

Table 1 represents a set of partial matrix of reviews. Each review $e_{ij} \in [1, 5]$ represents the score by a reviewer r_i on an entity e_j, '-' denotes that the reviewer did not give any score for that particular entity. For a given pair of entities (e_j,e_{j+1}), we count how many times entity e_j scores higher than entity e_{j+1}, how many times entity e_{j+1} scores higher than e_j, and how many times are equal. We discard any comparisons where there are no scores. For example, for the pair (e_1,e_2), we compare all scores that share in common. We have that reviewer r_1 scored entity e_2 higher than entity e_1, reviewer r_2 did not give any score to the entity e_1, thus we discard this comparison. Reviewer r_3 has the same situation as reviewer r_2, and reviewer r_4 scored entity e_2 higher than entity e_1. The number of cases where entity e_2 is higher than entity e_1 is two. The number of cases where entity e_1 is higher than entity e_2 is zero. Therefore, we generate the pair (e_2,e_1) in which entity e_2 is being preferred over entity e_1. We then compare entity e_1 with entity e_3 using the same procedure. We discover that the number of cases where entity e_1 is higher than entity e_3 is two, whereas entity e_3 is not being preferred over entity e_1. Thus, we generate the pair (e_1, e_3). For the next pair (e_2, e_3) the number of cases where entity e_2 is higher than entity e_3 is one and entity e_3 is higher than entity e_1 by one too. In this

particular case, entity e_2 and entity e_3 are treated as the equal. We will discuss this specific case in the next subsection.

In Figure 2 we describe the pseudo code for REVA evaluation. We start by selecting a pair of entities (*A,B);* counting the number of scores that entity *A* is higher than entity *B*, the number of scores that entity *B* is higher than entity *A*, and the number of scores that both entity *A* and *B* are equal. We do the same for all pairs in our target set. The output of this phase serves as input to the next phase for generating the preferred directed pair set.

3.2.2 Preferred Directed Pair Set

After counting all possible pairwise comparisons, we create the preferred directed pair set. The preferred directed pair set (*PDP*) stores the pairs where entity *x* is preferred over entity *y*. For example, for the pairwise comparisons of entity *x* and entity *y*, if entity *x* is being preferred over entity *y*, we add the pair (*x,y*) to the *PDP* set. After calculating the comparisons in question for each group of pairs, we add the preferred pair to the *PDP* set as follows:

1. Set constant *C* and threshold *T*. Let *C*=10, *T*=0.10
2. Calculate $N = \#(e_j,e_{j+1}) + \#(e_{j+1},e_j) + \#(e_j = e_{j+1})$
3. Calculate Adjusted Normalized Difference (*AND*) for each group of pairs (line 2)

   $AND = (\#(e_j, e_{j+1}) - \#(e_{j+1}, e_j)) / (N + C)$
4. If the number of reviews where (e_j,e_{j+1}) and *AND* is bigger than the threshold *T*, calculate its *difference average* which is $average(e_j) – average(e_{j+1})$ (it could be negative). Then, we add the pair (e_j, e_{j+1}), its *AND*, and its *difference average* to the *PDP* set.

5. Else if the number of reviews where (e_{j+1},e_j) and *AND* is bigger than the threshold *T*, calculate its *difference average* average (e_{j+1}) – average (e_j). We then add (e_{j+1}, e_j), its *AND,* and its *difference average* to the *PDP* set.

In these first five steps, we are taking into consideration the *AND* above the threshold *T*. As mentioned, we only need to calculate their *difference average* and add them to the *PDP* set. We are adding the *difference average* in order to break the ties and we will elaborate on that later.

We sometimes are presented with a case that may have the *AND* smaller than the threshold, this happens because #(e_j,e_{j+1}) and #(e_{j+1},e_j) could be very close e.g. #(e_j,e_{j+1}) = 100 and #(e_{j+1}, e_j) = 99. In real life, these two entities would have similar quality. This can make it difficult to select which entity is more likely to be preferred. For this reason, we have decided to set a threshold where we can identify such cases:

1. For the preferred pair (e_j,e_{j+1}) with the *AND* smaller than the threshold *T*, we calculate the *difference average* of average(e_j) – average(e_{j+1}) *da1* and *difference average* of average(e_{j+1}) – average(e_j) *da2*.
2. If *da1* is bigger than *da2* then we add (e_j,e_{j+1}) and its *difference average* of average (e_j) – average (e_{j+1}) . Else if *da2* is bigger than *da1* we add the pair (e_{j+1},e_j) and its *difference average* of average (e_{j+1}) – average (e_j) into the *PDP* set.
3. In case where both *difference averages* are equal, we make use of the total number of reviewers of each entity to break this tie. If the total number of reviewers from entity e_j is bigger than the total number of reviewers from e_{j+1} then we add (e_j,e_{j+1}) (line 10). Otherwise if the total number of reviewers from entity e_{j+1} is bigger than the total number of reviewers from e_j, we add (e_{j+1}, e_j) (line 12).

If still the total number of reviewers of both entities is equal, we randomly select a pair and add it to the *PDP* set. (line 14)

Algorithm: *REVA evaluation*

Input: Array of entities, List of ratings

Output: Array of ranked entities

```
1:  If(#(A,B)  ) then
2:      Calculate AND = #(A,B) - #(A,B) / ( N + C )
3:      if( AND < threshold )
4:        if (avg(A) – avg (B) > avg(B) – avg(A) )
5:            Add (A,B) and avg(A) – avg(B) into PDP set
6:        Else if (avg(A) – avg (B) < avg(B) – avg(A) )
7:            Add (B,A) and avg(B) – avg(A) into PDP set
8:        Else if (avg(A) – avg (B) = avg(B) – avg(A) )
9:            If(Number of Reviewers (A) > Number of Reviewers (B) )
10:               Add (A,B) into PDP set
11:           Else if(Number of Reviews (A) < Number of Reviews (B) )
12:               Add(B,A) into PDP set
13:           Else if(Number of Reviews (A) = Number of Reviews (B) )
14:               Random Add (A,B) or (B,A) into PDP set
15: If( #(A,B) equal #(B,A)) then
16:     We add the pair with bigger difference average.
17: if pairs never compare, add to PDP set as isolated or never compare
18:     Add (A,B) , AND = 0, AVG = 0
19: Order PDP set by AND, then by difference average
20: Create CP set base on PDP.
21: Create the final ranking set base on CP set and add isolated entities.
```

Figure 2: REVA evaluation pseudo code

3.2.3 Tie-breaking Schema

In the previous subsection, we have covered cases where the number of reviews for the pair (e_j,e_{j+1}) is bigger than the pair (e_{j+1}, e_j) and vice-versa. On the other hand, we have investigated cases where the number of reviews for the pair (e_j,e_{j+1}) is equal to the number of reviews of the pair (e_{j+1}, e_j). In regards to applying the same principle of *AND* under the threshold, we have designed a tie-breaking scheme to address this particular situation:

1. For example, if $\#(e_j,e_{j+1}) = \#(e_{j+1},e_j)$, to break this tie, we compute the average score of e_j from all the number of reviewers who have reviewed e_j and the average score of e_{j+1} from all the number of reviewers who have reviewed e_{j+1}. We add the pair (e_j,e_{j+1}) to the *PDP* set if $average(e_j) - average(e_{j+1})$ is bigger than $average(e_{j+1}) - average(e_j)$. If the opposite, we add the pair (e_{j+1}, e_j). Note that the *AND* will be equal to zero.
2. In case where both *difference averages* are equal $average(e_j) - average(e_{j+1}) = average(e_{j+1}) - average(e_j)$, we process the pair (e_j,e_{j+1}) if the number of reviewers who have reviewed e_j is bigger than the number of reviewers who have reviewed e_{j+1}; otherwise we process (e_{j+1},e_j) if the number of reviewers who have reviewed e_{j+1} is bigger than the number of reviewers who have reviewed e_j.

3.2.4 Isolated Entities

Often, a reviewer *r* might evaluate only one entity *e*, thus this entity *e* is never compare against others entities. This type of ideology is known as *isolated entities*. Isolated entities are ignored during the initial pairwise comparison and it is not to be added to the *PDP* set. However, they will be included in the final ranking set.

Table 2 represents a set of partial matrix from reviews. Reviewer r_5 only assigns a score of four to entity e_4. Using the pairwise comparison methodology, entity e_4 will not be compared to the rest of the entities; this will then lead to an *isolated entity*.

Table 2: Partial matrix with isolated entity

	r1	**r2**	**r3**	**r4**	**r5**	**r6**
e1	3	-	3	4	-	-
e2	4	2	-	5	-	3
e3	-	4	2	1	-	1
e4	-	-	-	-	**4**	-
e5	-	2	-	-	-	5

We should take special consideration in regard to isolated entities. There can always be a chance in which we are presented with a pair of entities that may never be compared. Table 2 represents the pair (e_1,e_5) where reviewer r_1,r_3 and r_4 evaluate entity e_1, whereas entity e_5 is evaluated by reviewer r_2 and r_6. In example, neither e_1 nor e_5 are considered isolated entities, because for the pair (e_1,e_2) they share reviews in common, as well for the pair (e_2,e_5). In order to tackle such cases, we must follow the same procedures that we have used for the *tie-breaking* scheme.

3.2.5 Ranking *PDP* Set

Having all preferred pairs computed and stored in the *PDP* set, we can start to order the *PDP* set by the *AND* scores and then followed by the *difference average* in descending order. When ordering preferred pairs by the *AND* and the *difference average*, we might encounter ties between them. For example, for the pairs (A,B) and (C,D) with

the same *AND,* we order (A,B) first if its *difference average* is bigger than (C,D) *difference average.* For this example, if still *AND* and *difference average* are equal, we order the pair (A,B) first if the sum of the number of reviewers who have reviewed entity *A* + the number of reviewers who have reviewed entity B (#(A) + #(B)) is bigger than the sum of the number of reviewers who have reviewed entity C + the number of reviewers who have reviewed entity D. If and when the opposite is presented, then we order the pair (C,D) before (A,B). If *AND*, *difference average* and the sum of the reviewers who have reviewed both entities of each pair are still equal, then we randomly order (A,B) or (C,D) first. By ordering the *PDP* set, we then ensure that the pairs of higher positions are more likely to be preferred. For instance, for a given pair (e_j,e_{j+1}) with the *AND* above the threshold and high *difference average* average(e_j) – average(e_{j+1}) will be ordered in a higher position in the *PDP* set. Therefore, we can assume that (e_j,e_{j+1}) is indeed strongly preferred among reviewers who have reviewed these two entities.

3.2.6 Compatible Directed Pair

The compatible directed pair set, better referred to as *CP,* is a set of pairs that must always maintain the transitive property. After creating the *PDP* set that contains the preferred pairs sorted by the *AND* scores, then by the *difference average* and then by the number of reviewers who reviewed both entities. CP set is derived by selecting all preferred directed pairs with the largest *AND* scores. Next, we process all pairs from the *PDP* set one by one in descending order. In the last step, we add the preferred directed pairs to *CP* if and when it does not violate the transitive property.

Transitive property: The transitive property or closure refers to a binary relation *R* over a set of X. if an entity *A* is related to entity *B*, and *B* is related to entity *C*, then *A* is also related to *C*.

The transitive property can be defined formally as:

$$\forall a, b, c \in X : (aRb \wedge bRc) \Rightarrow aRc$$

For example, if A>B and B>C, then A must be greater than C. If A<B and B<C, then C must be greater than A. Based on this property, if (A,B) and (B,C) are two preferred directed pairs in the *PDP* set with this order, we add (A,B) then (B,C) into the *CP* set. We can now infer that the next preferred directed pair must be (A,C).

Figure 3 represents the pseudo code for computing the compatible pair *CP* set which works as follows:

Algorithm: *Compute CP*

Input: Preferred directed pair PDP List

Output: Compatible directed pair CP List

```
1:  For i=0 i < sizeof(PDP) i++
2:      If Violation Found PDP(i)
3:          break and process next pair
4:      If new pair PDP(i) to be added is not in CP
5:          Add PDP(i) to CP
6:          Call InferNewPair(PDP(i), CP set)
7:          Add inferred new pairs into CP
8:      Else if new pair PDP(i) to be add is in CP
9:          break
10: Return CP set
```

Figure 3: Compute CP pseudo code

1. Once we have created the *PDP* set, this is passed as input to the Compute *CP* function. We process all preferred directed pairs in descending order (line 1).
2. If the preferred directed pair to be process is already in *CP*, then do nothing and continue processing the next one. In the case, that preferred directed pair we want to process could have already been inferred before (line 8).
3. Check if the preferred directed pair to be process violates the transitive property in *CP* set. For example, if (A,B) is the preferred directed pair to be process and (B,A) is already in *CP*, then (A,B) must be ignored and continue processing the next pair. If (C,A) is the preferred directed pair to be process and (A,B), (B,C) are already in *CP*, then (C,A) violates the transitive property and must be ignored. Continue processing the next pair (line 3).
4. If the next preferred directed pair to be process is not in *CP*, then we add it to *CP* set and call the *inferNewPair* function sending the preferred directed pair and the *CP* set as an input. Figure 4 shows the pseudo code for inferring new pair function which works as follows. First, we find pairs in *CP* that contain one of the same entities as the preferred directed pair that have already been added to *CP*. For example, if (A,B) was the preferred directed pair added to *CP*, then find any pair in *CP* that contains entity A or B. For instance (B,C), (D,A). If (B,C) and (D,A) are found in *CP*, then we can infer **(A,C)** , **(D,B)**, and **(D,C)**. In order to maintain the transitive property, we must check that they do not violate the transitive property and add them to inferred temp set. Each of those newly inferred pairs is sent to the *inferNewPair* function and performs the same procedure of trying to infer new pairs recursively (line 5).

Algorithm: *Infer new pair*

Input: Preferred directed pair and CP set

Output: Compatible directed pair

```
1:  If pairs found in CP with one of the same entities as the new pair to be add then
2:      For i=0 i<number of pairs found in CP i++
3:          If new inferred pair has no violations then
4:              Add it to Inferred Temp set
5:              Call inferNewPair(new Inferred pair, CP set)
6:          Else if new inferred pair has violations then
7:              Discard and continue inferring
8:  Else if no pairs found in CP with one of the same entities
9:      Break
10: Return Inferred Temp set
```

Figure 4: Infer new pair pseudo code

3.2.7 Computing Final Ranking List

After processing all preferred directed pairs from the *PDP* set into the Compatible directed pair set, we then compute the final ranking list. Figure 5 represents the pseudo code for computing the final ranking list. For a given set of pairs {(A,B), (B,C), (A,C), (D,C)} in the *CP* set, we process each pair at a time in descending order. We first process the pair (A,B) and then we generate the final ranking list {A,B}. The next pair is (B,C) and process it {A,B,C}. The third pair to be processed is (A,C) which we do not do anything since we have both entities in the final ranking list already. The final pair to be processed is (D,C). For this pair, we add entity D before entity C and which gives us the final ranking set for that particular *CP* set: {A,B,D,C}.

Adding isolated entities: We have come up with the final ranking list; however, we still need to process the isolated entities. The isolated entities were removed at the pairwise

comparison step; these entities are added to the final ranking set using the following algorithm:

1. Calculate isolate entity average from all reviewers who reviewed the entity.
2. In the final ranking list, we find the entity *e* with the lowest position with an average greater than the average of the isolated entity.
3. Insert the isolated entity below the entity *e*
4. If the isolated entity average is greater than all entities in the final ranking list, then add the isolated entity at the beginning of the final ranking list.

For example, for the final ranking list {A, B, D, C} with an average of 4.3, 3.5, 3.7, 2.5 respectively for each entity, and an isolated entity E with an average of 3.6. The entity with the lowest position with an average greater than the isolated entity E will be entity D. The Isolated entity E must be inserted after entity D: {A, B, D, **E**, C}.

Algorithm: *Compute final ranking list*

Input: CP set

Output: Final ranking list

```
1:  For i=0 sizeof(CP) i++
2:      If CP(i) left side entity is not in the final ranking list then
3:          Add left side entity to the final ranking list
4:          If CP(i) right side entity is in the final ranking list then
5:              Move left side entity in front of right side entity
6:      If CP(i) right side entity is not in the final ranking list then
7:          Add right side entity to the final ranking list
8:      If CP(i) right side entity && left side entity are in the final ranking list  && right
        side entity position > left side entity position then
9:          Move left side entity in front of right side entity
10: Return final ranking list
```

Figure 5: Compute final ranking list pseudo code

3.3 Example

For demonstration purposes, an example of REVA evaluation will be presented to cover all possible scenarios. We are going to use a sample with eleven entities and eleven evaluators containing partial ratings.

Table 3: Sample eleven entities and eleven evaluators

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}
e_1	2							5			
e_2					1						
e_3	2				1			2		2	
e_4	3			2	5		1			5	
e_5		4					2			3	
e_6						5	1				
e_7		1	1		1		5		1	5	
e_8										2	
e_9							1				
e_{10}						5					
e_{11}											3

Table 3 shows an extreme sparse matrix with eleven entities and eleven reviewers. The first step is to compute all pairwise comparisons; using the formula $\frac{1}{2}$ N (N-1), which will lead us to have fifty-five comparisons. After all comparisons are made, we will then select the pair with the highest number of preferred ratings, as well as calculate the difference average and the *adjusted normalized difference*. From the table, we observe that entity e_{11} is an isolated entity because there is no comparison with the rest of the entities. We will ignore this entity during the pairwise comparison stage and add it to the final ranking list.

Table 4: Preferred directed pair set example

Pairs	*AND*	Difference average	Pair	*AND*	Difference average
(e_4,e_3)	0.230	1.450	(e_5,e_2)	0	2
(e_4,e_2)	0.09	2.200	(e_{10}, e_5)	0	2
(e_5,e_9)	0.09	2	(e_6,e_9)	0	2
(e_7,e_9)	0.09	1.333	(e_{10},e_6)	0	2
(e_5,e_3)	0.09	1.25	(e_6,e_2)	0	2
(e_4,e_8)	0.09	1.200	(e_{10},e_4)	0	1.799
(e_5,e_8)	0.09	1	(e_1,e_8)	0	1.5
(e_7,e_8)	0.09	0.333	(e_{10},e_1)	0	1.5
(e_5,e_6)	0.09	0	(e_7,e_2)	0	1.333
(e_4,e_1)	0.09	-0.29	(e_6,e_3)	0	1.25
(e_7,e_6)	0.09	-0.66	(e_1,e_7)	0	1.16
(e_1,e_3)	0.083	1.75	(e_6,e_8)	0	1
(e_7,e_3)	0.083	0.583	(e_8,e_2)	0	1
(e_5,e_7)	0.076	0.666	(e_8,e_9)	0	1
(e_{10},e_2)	0	4	(e_4,e_7)	0	0.866
(e_{10}, e_9)	0	4	(e_3,e_2)	0	0.75
(e_{10},e_3)	0	3.25	(e_3,e_9)	0	0.75
(e_{10}, e_8)	0	3	(e_1,e_5)	0	0.5
(e_{10},e_7)	0	2.666	(e_1,e_6)	0	0.5
(e_1,e_2)	0	2.5	(e_8,e_3)	0	0.25
(e_1,e_9)	0	2.5	(e_4,e_5)	0	0.200
(e_4,e_9)	0	2.200	(e_4,e_6)	0	0.200
			(e_2,e_9)	0	0

Table 4 represents the *PDP* set which includes all preferred pairs ordered by the *adjusted normalized difference* and then by the difference average in descending order. Next, we will process all preferred pairs one by one into the Compatible Directed Pair *CP* set in order to maintain the transitive property.

CP = { (e_4,e_3), (e_4,e_2), (e_5,e_9), (e_7,e_9), (e_5,e_3), (e_4,e_8), (e_5,e_8), (e_7,e_8), (e_5,e_6), (e_4,e_1), (e_7,e_6), (e_1,e_3), (e_7,e_3), (e_5,e_7), (e_{10},e_2), (e_{10},e_9), (e_{10},e_3), (e_{10},e_8), (e_{10},e_7), (e_{10},e_6), (e_1,e_2), (e_1,e_9), (e_4,e_9), (e_5,e_2), (e_{10},e_5), (e_6,e_9), (e_6,e_2), (e_7,e_2), (e_{10},e_4), (e_{10},e_1), (e_1,e_8), (e_6,e_3), (e_1,e_7), (e_4,e_7), (e_4,e_6), (e_1,e_6), (e_6,e_8), (e_8,e_2), (e_8,e_9), (e_3,e_2), (e_3,e_9), (e_1,e_5), (e_4,e_5), (e8,e_3), (e_2,e_9)}

From the CP set, we will process each pair one by one and create the final ranking list. Table 5 shows the final ranking list:

Table 5: Final ranking list

Rank	Average
e_{10}	5
e_4	3.2
e_1	3.5
e_5	3
e_7	2.333333
e_6	3
e_8	2
e_3	1.75
e_2	1
e_9	1

The final step would be to add all the isolated entities. For this particular sample, entity e_{11} is an isolated entity with an average of three. Next, we find the entity with the lowest position that has the average greater or equal to the isolated entity and add the isolated entity after it. Next, we present the final REVA ranking list with the isolated entities, average ranking and true ranking.

Table 6: REVA ranking vs. average ranking

REVA	Average
e_{10}	5
e_4	3.2
e_1	3.5
e_5	3
e_7	2.33
e_6	3
e_{11}	3
e_8	2
e_3	1.75
e_2	1
e_9	1

Average	Average
e_{10}	5
e_1	3.5
e_4	3.2
e_5	3
e_6	3
e_{11}	3
e_7	2.33
e_8	2
e_3	1.75
e_2	1
e_9	1

True Ranking	Average
e_2	3.36
e_5	3.28
e_{10}	3.25
e_4	3.22
e_3	3.18
e_8	2.99
e_1	2.97
e_7	2.93
e_6	2.91
e_{11}	2.85
e_9	2.46

From Table 6, we can observe that REVA ranking entity e_4 was ranked higher than entity e_1 even though entity e_1 has a bigger average than entity e_4. The same case applies for entity e_7 which was ranked higher than entity e_6 and e_{11}. If we compare this observation against the true ranking, we can determine that indeed entity e_4 has better quality than entity e_1 and, entity e_7 has better quality than entity e6 and e_{11} as well. However, based on the average rating, entity e_1 is ranked higher than entity e_4 and entity e_7 is ranked higher than entity e_6 and e_{11} which is proven not to be true.

Error rate: Table 7 shows the error rate of REVA evaluation and average rating against the true ranking using Kendall tau distance and weighted Kendall tau distance from this example.

Table 7: REVA vs. average rating error rate example

	REVA evaluation	**Average rating**
Kendall Tau Distance	0.381818	0.436363
Weighted Kendall Tau Distance	0.28523	0.307338

4. REVA EVALUATION ASSIGNMENT

In REVA project, we also investigate a closely related topic of evaluation assignment. In this section, we will further discuss improvements that REVA evaluation can apply. We will now explain the main ideas and methodologies.

4.1 REVA Evaluation Assignment Overview

Before evaluation is conducted there is an assignment phase that assigns target entities to evaluators, it is common to assign these target entities to the evaluators randomly. To the best of our knowledge, currently, none of the evaluation systems design evaluator assignment to maximize evaluation quality. Without intelligent assignment, a lot of evaluation effort would be wasted on obvious/easy comparisons (e.g., to compare two items with a big difference in quality). In real scenarios such as academic conferences, each paper has to be reviewed by five different evaluators. The assignment is made arbitrarily such as each evaluator reviews five papers randomly. We propose a *layered assignment* approach to maximize the quality of comparison-based evaluation for given evaluation resources, leading to improved approximation to ground truth.

The main idea of REVA evaluation assignment is to perform evaluation layer by layer. In each layer, we assign several comparable entities to each available evaluator where entities are comparable if they receive similar evaluations from the previous round. The assignment of comparable entities to the evaluators leads to maximized total ranking improvement from layer to layer. Moreover, the total ranking improvement also leads to better assignment from layer to layer because the compatibility of the comparable entities will be adjusted and gradually improved along the way. For a better performance and evaluation quality, it is desirable to use as many layers as possible. REVA evaluation

assignment approach makes use of the existing conventional single-layer REVA evaluation approach.

4.2 REVA Evaluation Assignment Methodology

In this section we will explain the main concepts and methodologies for the REVA evaluation assignment approach. We discuss its different phases such as layering, assignments and ranking.

4.2.1 Layering Phase

The main idea of the REVA evaluation assignment is to perform evaluations layer by layer. In order to achieve this task, we need to manually calculate/set the number of layers according to the number of entities to be evaluated, and the number of available evaluators. For each layer, we assign a percentage of the available evaluators. A good example would be to set four layers for a given sample of one hundred entities and one hundred evaluators; which gives us a total of twenty-five evaluators for each layer if we assign 25% of the evaluators to each layer. Let assume that each evaluator must evaluate four different entities and each entity has to be evaluated by four different evaluators. The assignment would be best represented on Table 8:

Table 8: Assignment representation

Layer	Number of entities	Number of evaluators	Number of evaluations
1	100	25	25x4=100
2	100	25	25x4=100
3	100	25	25x4=100
4	100	25	25x4=100

For the first layer, we make use of the one hundred entities and randomly choose twenty-five evaluators. In the second layer, we choose the next twenty-five evaluators and do the same for layer three and layer four. In each layer, each evaluator will evaluate an average of four entities and each entity has to be evaluated at least once.

For each layer, a ranking list will be generated and then used by the next layer. This leads to having a better compatibility and comparability. Next, we will describe the procedure for entities assignment in each layer.

4.2.2 Assignment Phase

In this phase, we propose an efficient solution for entities assignment. Having a random assignment or letting the users to select entities to evaluate leads bad comparability between entities. Table 9 is a representation of a bad assignment. From entity 1 to entity 4, they share reviews in common and from entity 5 to entity 8 as well. However, there are no reviews between these two groups of entities. For example, there is no comparison or reviews in common between entity 1 and entity 5. Therefore, there will be poor comparability that will result in a bad quality ranking.

Table 9: Bad assignment example

	R_1	R_2	R_3	R_4
E_1	1	1	-	-
E_2	1	1	-	-
E_3	1	1	-	-
E_4	1	1	-	-
E_5	-	-	1	1
E_6	-	-	1	1
E_7	-	-	1	1
E_8	-	-	1	1

In order to achieve a better comparability, we need to ensure that all entity have the capability to be connected. To make sure that every entity is connected to each other, the assignment of each evaluator needs to overlap and as even as possible. Figure 6, we demonstrate how the assignment should be made.

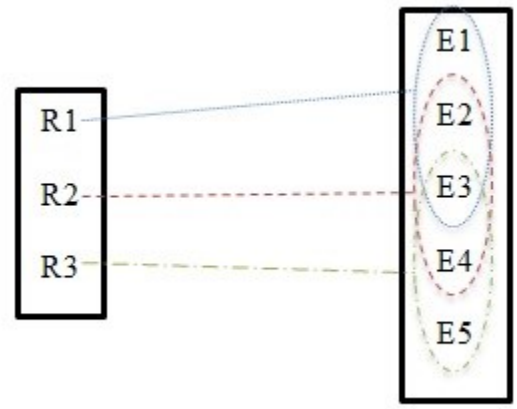

Figure 6: Assignment phase

For example, for entities E_1, E_2, E_3, E_4, E_5 and reviewers R_1, R_2, R_3 we assign { E_1,E_2,E_3} to R_1, { E_2, E_3,E_4} to R_2 and { E_3, E_4, E_5} to R_3 as shown in Figure 6. This assignment ensures that more comparisons can be made between these entities.

Table 10 shows a good assignment example. In this example, all entities share some common reviewers. In other words, all entities are connected. We then can ensure that comparisons can be made between all entities.

Table 10: Good assignment example

	R_1	R_2	R_3	R_4	R_5
E_1	1	-	-	-	1
E_2	1	-	-	-	1
E_3	1	1	-	-	-
E_4	1	1	-	-	-
E_5	-	1	1	-	-
E_6	-	1	1	-	
E_7	-	-	1	1	
E_8	-	-	1	1	
E_9	-	-	-	1	1
E_{10}	-	-	-	1	1

4.3 Example

The following example demonstrates REVA evaluation assignment process. Suppose we have one hundred entities and one hundred evaluators. We have the constraint that each evaluator can only evaluate ten different entities.

Layering: Let the number of layer = 3. For the first layer, we randomly select 40% of the evaluators and assign ten entities to each evaluator. In the second layer, we select 30% from the rest of the evaluators and assign ten entities to each evaluator. For the third layer, we select the remaining 30% of the evaluators and assign them ten entities to evaluate.

Table 11: Layer assignment example

	Evaluators	Number of evaluations
Layer 1	40%	40x10 = 400
Layer 2	30%	30x10 = 300
Layer 3	30%	30x10 = 300
Total	100%	1000

From Table 11, we can observe the evaluation distribution. There will be a total of one thousand evaluations. Because we do not have any prior information about the entities, we have decided to set a high percentage of evaluators to layer one so we can get sufficient information for further layers.

Assignments: The next step is to assign entities to every evaluator. In total, each entity would receive an average of ten evaluations. For the first layer, we randomly select 40% of the evaluators and assign ten entities to each of them. In this layer, each entity will receive an average of four evaluations. Next, we randomly select 30% from the rest of the evaluators. This would result in thirty evaluators. We assign ten entities to each evaluator having three hundred evaluations. Each entity would receive an average of three evaluations. In the second layer, the entities will be ranked based on the layer one evaluations. This will lead to having similar quality entities close to each other. The assignment will overlap and be as even as possible as shown in Figure 6. After the assignment and the evaluation are performed, we will run REVA evaluation algorithm for the layer two. The result will be an improvement of the ranking from layer one, leading it to have similar quality entities justified and closer to the true ranking.

In layer three, we randomly select the remaining 30% of the evaluators. As in layer two, we assign ten entities to each evaluator. We also run REVA evaluation algorithm on this assignment and evaluation. The final result should be a ranking list that is better approximated to the true ranking.

5. IMPLEMENTATION AND DEMONSTRATION

In this section, we will discuss the technology and technical aspects in details of REVA implementation. Both REVA evaluation and REVA evaluation assignment frameworks were implemented on Visual Studio Ultimate 2012 using C sharp (C#) as a programming language.

5.1 Database

Our study consists of numerous experiments and comparisons over different datasets. Therefore, we must ensure that we are using the right tools that support large data. The database management system used in our study is Microsoft SQL Server 2008. This DBMS stores the different datasets and customized subsets used in our experiments.

5.2 Architecture Design

Figure 7 shows the main architecture of REVA evaluation framework. REVA evaluation makes use of customized ratings datasets stored in MS SQL Server. The processing data module connects to the database to extract the data, then manipulates and organizes the data into list collections that are sent to the main algorithm.

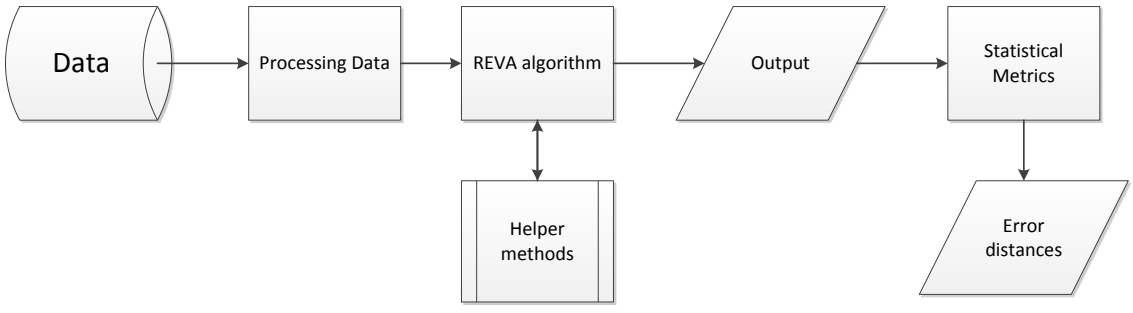

Figure 7: REVA evaluation design

The algorithm takes an array of entities and a list of all ratings as an input parameters and a final ranking list with its average as an output. The algorithm makes use of different helper functions in order to make the framework modular, easy to understand and debug.

The statistical metrics module computes the error distance between two ranking lists. This module makes use of well-known statistical methods such as Kendall Tau distance and its different variations: normalized, correlation, and weighted. We compare our final ranking list from REVA evaluation and average ranking list against the true ranking. The expected result from this module is to show that our ranking list is closer to the true ranking.

5.3 Technology

The REVA project consists of different components such as REVA evaluation and REVA evaluation assignment algorithms, error distance metrics functions, and demo implementation. The REVA project was implemented in .NET Framework 4.5 using Visual Studio 2012 as an IDE (integrated development environment). The REVA evaluation algorithm is implemented as a class library where it can be consumed by any other project as a reference. The REVA evaluation class library includes the following important methods and structs:

- DirectedPair struct
- RankingList struct
- List<RankingList> evaluation(int[ ] EntityArray, List<int[,]> Ratings, float constant, float threshold)

The Next component of the REVA project is the error distance metrics. This is also implemented as a class library that is consumed to measure two ranking lists. The Error distance metrics project includes the following methods:

- Double KendallNormalized(int [ ] listOne, int[ ] listTwo) – This method computes the Kendall tau normalized distance in the range of [0,1]
- Double WeightedKendallNormalized(IComparable[,] listOne, IComparable[,] listTwo) – This method computes the Weighted Kendall Tau Normalized Distance in the range of [0,1]
- Double KendallCorrelation(int [ ] listOne, int[ ] listTwo) – This method computes the Kendall Tau correlation distance in the range of [-1,1]

5.4 Demo Implementation

We have implemented a web-based demo application for the REVA project built in ASP.NET web forms with .NET framework 4.5. The main purpose of the demo application is to demonstrate the advantages of REVA evaluation and REVA evaluation assignment. For data extraction, the demo application utilizes databases stored in Microsoft SQL Server.

Functionality: The demo application extracts a partial sample with sparse data from a dataset. Then, REVA evaluation and average rating are computed from this sample. The true ranking is computed from the dataset that is a representation of the "whole" population ratings. After that, we calculate the error rate of REVA and average rating against the true ranking using Kendall Tau distance and Weighted Kendall Tau distance. Finally, we display all three ranking list, the error rate, and the partial sample data. We use the same procedure for REVA evaluation assignment.

Figure 8: REVA evaluation project demo

In Figure 8, we present the REVA evaluation demo homepage. By clicking on the "Run Sample" button, the application generates/extracts a subset/sample from a database in Microsoft SQL Server. Then, it displays the generated data (Figure 9), REVA evaluation, average rating and true ranking.

Data

movieid	109	301	410	480	499	556	643	668	713	806
1	4	4					5			4
22	4	4								
25	4	3								
64	2	5		3		5				
69	4	5			5		3	1		
82	5	5					3	4		4
96	5	5		4			5			5
97	3	4			4			2		
111	4	1					4			3
132		4			4	5	5			
143		4			3		4			5
151	5	2								
173	5	4			5	3	4			
185				2			5			
195	5	5					5			3
215	3	5			4		3			
235		2					4			
237	4	4		2						2

Figure 9: REVA evaluation demo data

It also demonstrates the Kendall tau and weighted Kendall tau error rate of REVA evaluation and average rating against the true ranking as shown in Figure 10.

Error rate

Rank	Kendall Tau Distance Normalized	Weighted Kendall Tau Distance
REVA ranking	0.246305418719212	0.140301433647837
Average ranking	0.293103448275862	0.191520646979537

Figure 10: REVA evaluation demo error rate

Object relational mapping: For object relational mapping, our demo application uses Simple.Data[1] framework. Simple.Data is a lightweight framework that uses the dynamic features of .NET 4 to offer an expressive, ORM stylish way of accessing and manipulating data. This framework allows faster data manipulation by eliminating the need to write SQL queries for data retrieval.

[1] http://simplefx.org/simpledata/docs/index.html

6. EXPERIMENTS

The objective of experiments on real-life datasets is to prove the efficiency of the proposed REVA evaluation and REVA evaluation assignment, principally by comparing them against average rating. Then, using several case examples with different data size, we conduct an overall comparison between both REVA approaches and average rating against true ranking. Our results show that REVA evaluation and REVA evaluation assignment perform better than average rating showing a closer similarity with the true ranking. The experiments were conducted on a PC Intel i5 M 460 @ 2.53GHz CPU with 6 GB of RAM, running Windows 7 Professional.

6.1 Datasets

The datasets used in these experiments were collected from the social computing research at the University of Minnesota (GroupLens) and Yahoo! Webscope. Specifically, we used the "MovieLens 100k[2]" which contains ratings by users of the movie recommendations site MovieLens. This MovieLens 100k dataset consists of 100,000 ratings (1-5) from 943 users on 1682 movies. Each reviewer has rated at least twenty movies and each movie may be reviewed by at least one reviewer. We also used the Yahoo! Webscope dataset ydata-ymusic-user-artist-ratings-v1_0[3]. This dataset contains 15,400 users, 1000 songs and approximately 300,000 ratings.

[2] http://grouplens.org/datasets/movielens/
[3] http://research.yahoo.com/Academic_Relations

6.2 Error Rate vs. Sample Size

As we described in previous chapters, according to the law of large numbers, average ratings from a large sample can well approximate the true rating and true ranking. However, obtaining an average rating from a small subset differs significantly from the true ranking. To confirm such statement, we conduct an experiment using small subsets from the whole MovieLens dataset. For each small subset, we calculate the average rating, rank them and compare the similarity with the true ranking. We compare the ranking lists using Kendall Tau distance (Kendall & Gibbons, 1990). Given two ranked list X and Y, Kendall tau distance counts the number of pairwise disagreements between X and Y. Suppose, we have the following two lists:

$$X = \{A,B,C,D\}$$

$$Y = \{B,A,D,C\}$$

Rank list X contains a set of entities A, B, C, D in that particular order, whereas rank list Y contains B, A, D, C in that particular order. Table 12 represents the number of discordant and concordant pairs. The Kendall tau distance between rank list X and Y is two. Usually, Kendall tau distance is normalized by the following formula, where n is the size of X and Y:

$$Kendall(\mathrm{X},\mathrm{Y}) = \frac{Discordant\ pairs}{\frac{1}{2}n(n-1)}$$

Table 12: Pairwise disagreement counting

Pair	X	Y	Count
(A,B)	A>B	A<B	✗
(A,C)	A>C	A>C	✓
(A,D)	A>D	A>D	✓
(B,C)	B>C	B>C	✓
(B,D)	B>D	B>D	✓
(C,D)	C>D	C<D	✗

According to the formula, Kendall(X,Y) will be equal to zero if the two lists are identical and one if X is in the reverse order of Y. Then, the normalized Kendall tau distance lies in the range [0,1]. For this particular example, the normalized Kendall tau distance will be 0.33.

Figure 1 represents a graph of Kendall similarity between average rating and true ranking for different number of reviews. In this particular example, we first extract one review/rating for each entity and take the average; then we continue increasing the number of reviews and computing the average. For each ranking list, we compute the Kendall tau distance against the true ranking. Our observation confirms that where there are many reviews/ratings per entity, the error distance decreases. In other words, with a large number of ratings, average rating gets close to the true ranking. On the other hand, with a small number of ratings, the error distance increases.

6.3 Brute Force Approach on Small Data

We have implemented a brute force rank aggregation algorithm to compute a ranking list and compare it against REVA evaluation and average rating. This approach should perform well and possibly better than average rating in some scenarios.

As discussed in section 2.3, rank aggregation consists of combining ranking results of target entities from multiple ranking functions in order to generate a better one. One straightforward solution for rank aggregation is to make use of the old-fashion brute force approach. This approach works for small-size problems and it is not recommended when using $n > 10$.

The brute force rank aggregation algorithm works as follows: First, for a given list of entities, we calculate all possible permutation. For a given list of user's ratings, we rank each user's ranking list by the average. For each permutation, we compute the number of violations (discordant pairs) with respect to each user's ranking list. After that, we select the permutation ranking list with the smallest number of violations. In case we encounter more than one permutation ranking list with the same number of violations, we compute the Kendall tau distance for each permutation ranking list against average rating. Then, we select the permutation ranking list with the smallest error distance.

As we mentioned previously, the brute force approach only works well on problems with small number of entities. The time complexity for our brute force approach takes factorial time $O(n!)$ leading to poor performance with a larger number of entities. For instance, we require 10! (3628800) number of permutation to evaluate a ranking list with size ten.

We conduct experiments comparing the performance of brute force aggregation approach, average rating and REVA evaluation against true ranking. We have created ten different subsets from Yahoo! Music dataset consisting of ten entities and ten users having extreme sparse data with an average of twenty-five ratings.

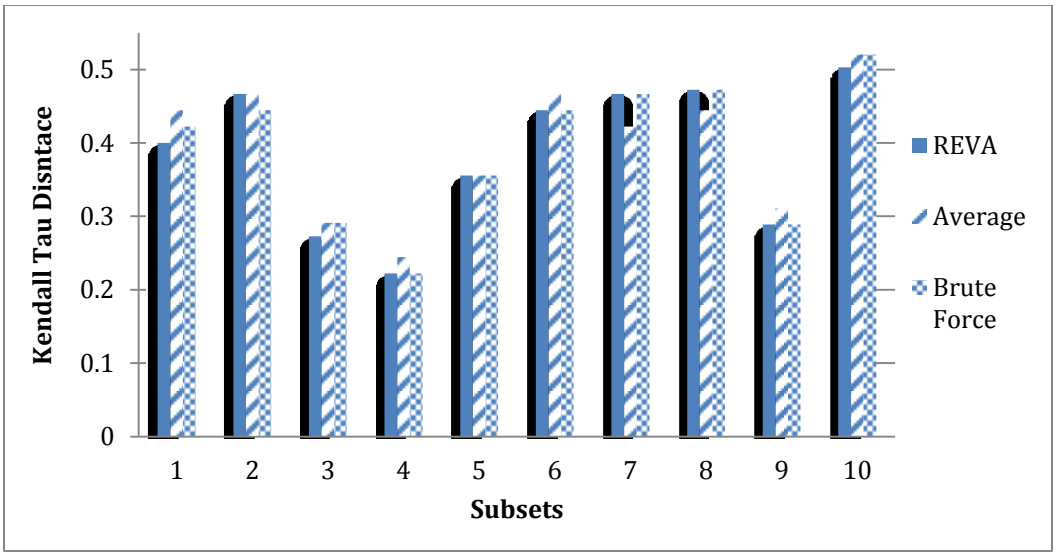

Figure 11: Bruce force rank aggregation vs. REVA and average rating

Figure 11 represents a benchmarking between brute force rank aggregation approach, REVA evaluation and average rating. Brute force approach performed better than REVA evaluation and average rating in one occasion (subset 2). It also performed the same as average rating in four occasions whereas REVA evaluation obtained smaller error distance than the brute force approach and average rating in five occasions. Figure 12 shows the error rate averages from the ten subsets.

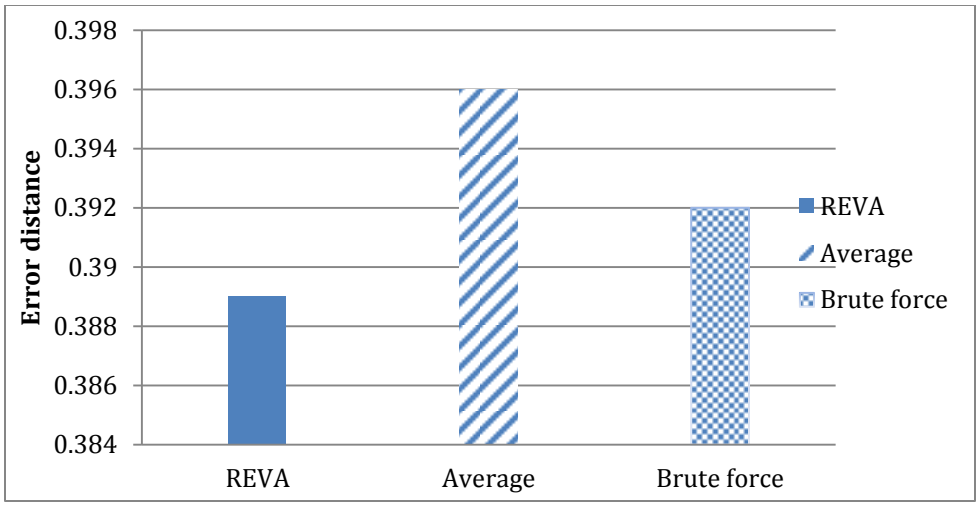

Figure 12: Brute force error rate average

As mentioned before, brute force approach does not perform well with large number of entities; therefore, it will be almost impossible to continue showing comparisons since we are going to use larger datasets in further experiments.

6.4 Experiments for REVA Evaluation

Having demonstrating the effects of small number of ratings on average ratings, here we conduct several experiments to study and demonstrate that REVA evaluation will perform better than average rating on small samples where there are few ratings per target entity.

First, we compare the quality of the rank list generated by REVA evaluation and average rating with respect to the true ranking. Given a generated rank list from REVA evaluation and average rating, in order to measure the outcome performance, we compute the Kendall tau distance explained in the previous section, and weighted Kendall tau distance against true ranking. A similarity value of zero specifies a complete agreement

between the two lists. Whereas a similarity value of one indicates a total disagreement between them.

We have created twenty different subsets from MovieLens and Yahoo! Webscope Music to observe the performance of REVA evaluation on different scenarios comparing to average rating. First, we extract thirty random entities and ten random reviewers from the whole MovieLens and Yahoo! Webscope Music datasets.

For each subset, we compute the ranking using REVA evaluation and average rating and measure the error distance against the true ranking using Kendall tau distance and weighed Kendall tau distance. Our main objective is to perform better than average rating in most of the cases using small subsets. As we defined in the law of large numbers, average rating well serves the purpose of approximate to true ranking using large number of reviews. We expect that REVA evaluation performs at least the same or close to average rating on large number of reviews. However, in reality, each reviewer typically reviews very few entities generating an extremely sparse data making it difficult for average rating to meet its purpose.

We obtained an error rate average of 0.31 from REVA evaluation whereas average rating obtained 0.33 using MovieLens dataset. During the experiments on the twenty different subsets, REVA evaluation performed eleven times better than average rating, two times both tied, and seven times average rating performed better than REVA evaluation.

We observed the error rate average between REVA evaluation and average rating against true ranking using Yahoo! Webscope dataset ydata-ymusic-user-artist-ratings-

v1_0. REVA evaluation obtained a smaller error rate average of 0.37 while average rating obtained 0.38.

6.5 Experiments for REVA Evaluation Assignment

We have conducted some experiments that demonstrate the performance of REVA evaluation assignment against random assignment using average rating and single-layer REVA evaluation approach.

First, we have created a total number of five samples with fifty entities and fifty evaluators each. For REVA evaluation assignment, each evaluator will evaluate ten entities and each entity will receive an average of ten reviews. We have set three layers with the following evaluators' percentage: 40% for layer one, 30% for layer two and the remaining 30% for layer three. For the first layer, we will have twenty evaluators and each entity an average of four reviews. For the second layer, we will have fifteen evaluators and the remaining fifteen evaluators for the third layer with an average of three reviews for each entity. We rank the entities using REVA evaluation algorithm during the evaluation process in each layer. We will come up with the final ranking in the third layer.

REVA evaluation assignment will be compared against two different approaches: Random assignment using average rating and single-layer REVA evaluation approach. For the random assignment, we use the same idea as the bad assignment example in Table 9. Each evaluator will evaluate ten entities, and each entity will receive ten reviews. After the random assignment is completed, we rank the entities base on average ratings. Then, we use the same random assignment evaluation to rank the entities using the single-layer REVA evaluation approach.

Then, we will calculate the error rate of REVA evaluation assignment, random assignment average, and random assignment single-layer REVA approach against true ranking using Kendall tau distance. The results show that REVA evaluation assignment obtained 0.3781, random assignment average obtained 0.3982 and single-layer REVA evaluation approach obtained 0.3985. We observe that REVA evaluation assignment obtained a smaller error rate comparing with the two other approaches. Due to bad assignment, single-layer REVA evaluation approach performed very similar to random assignment.

6.6 Discussion

Test results show that REVA evaluation and REVA evaluation assignment indeed performs better than average rating and brute force aggregation against true ranking. Although an average computer was used to perform the experiments, both REVA evaluation approaches performed relatively fast on large datasets. One limitation on these experiments is that brute force approach performed slowly; therefore it was almost impossible to conduct comparisons on larger datasets.

7. CONCLUSION

Average ratings used in conventional evaluation suffer from evaluation bias for sparse data. Based on the observation that comparative evaluations are more trustworthy than isolated ratings, in this study we have investigated comparison-based evaluation, targeting effective elimination of evaluation bias and improved approximation to true rankings and true ratings. We have also investigated an associated topic of evaluation assignment and proposed layered assignment, aiming at optimizing evaluation quality for given resources. We have implemented and validated the proposed algorithms using benchmark datasets in comparison with state-of-the-art methods. In addition, to demonstrate the utility of our approach, a prototype system has been deployed and made available for convenient public access.

There are many interesting directions for future work. Firstly, our proposed evaluation and assignment algorithms can be optimized in many ways, for example, by designing more sophisticated pair comparison, pair processing ordering, tie-breaking and inference schemes. Secondly, it would be interesting to perform a realistic case study, for example, conference paper evaluation, to further validate the proposed approaches. Thirdly, in this study we have assumed all ranks in the total ranking are equally important. In reality, there are cases where only part of the total ranking is important, or it is only important to find a rank-based cut-off point instead of ranking all the target entities. These preferences would require modifications and adaptations of both our proposed evaluation and assignment approaches. Fourthly, in the evaluation assignment problem we study, we have assumed that all evaluators have the same evaluation capacity and match the target entities equally. Both may not be true in reality. Ideally evaluator

capacity, expertise, and interests should be considered as additional constraints in the assignment process. Last but not least, our proposed approach can possibly be extended and used in applications beyond evaluation, such as recommendation. It is interesting to explore these possibilities.

8. BIBLIOGRAPHY

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng*, 734–749.

Arkes, H. (2003). The nonuse of psychological research at two federal agencies. *Psychol. Science*, 1–6.

Aslam, J., & Montague, M. (2001). Models for metasearch. *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Belkin, N., & Croft, W. (1992). Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 29-38.

Bell, R., & Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. *In Proceedings of the 2007 Seventh IEEE International Conference on Data Mining (ICDM)*.

Borda, J. (1781). Mémoire sur les élections au scrutin. *Historie de l'academie royale des sciences, Paris*.

Breitenbach, M., & Grudic, G. (2005). Clustering through ranking on manifolds. *In Proceedings of the 22nd International Conference on Machine Learning (ICML)*.

Chaudhuri, S., & Das, G. (2003). Automated ranking of database query result. *In Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR)*.

Chaudhuri, S., Das, G., Hristidis, V., & Weikum, G. (2004). Probabilistic ranking of database query results. *In Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*.

Chen, M., & Singh, J. (2001). Computing and using reputations for internet ratings. *In Proceedings of the 3rd ACM Conference on Electronic Commerce (EC)*.

Chen, W., Jianhua, X., Huai, L., Yue, W., Ming, Z., Eric, H., & Robert, C. (2010). Knowledge-guided gene ranking by coordinative component analysis. *BMC Bioinform*.

Collins, M. (2005). Discriminative reranking for natural language parsing. *Comput. Linguist*, 25–70.

Conry, D., Koren, Y., & Ramakrishnan, N. (2009). Recommender systems for the conference paper assignment problem. *In Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys)*.

Diaconis, P. (1988). Group representations in probability and statistics. *In Institute of Mathematical Statistics Lecture Notes–Monograph Series*, 11.

Dumais, S., & Nielsen, J. (1992). Automating the assignment of submitted manuscripts to reviewers. *In Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Dwork, C., Kumar, R., Naor, M., & Sivakumar, D. (2001). Rank aggregation methods for the web. *In Proceedings of the 10th International World Wide Web Conference (WWW)*.

Fagin, R., Kumar, R., & Sivakumar, D. (2003). Comparing top k lists. *In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Fagin, R., Kumar, R., & Sivakumar, D. (2003). Efficient similarity search and classification via rank aggregation. *In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD)*.

Ferndandez, M., Vallet, D., & Castells, P. (2006). Probabilistic score normalization for rank aggregation. *In Proceedings of the 28th European Conference on Advances in Information Retrieval (ECIR)*.

Furlanello, C., Serafini, M., Merler, S., & Jurman, G. (2010). Entropy-based gene ranking without selection bias for the predictive classification of microarray data. *BMC Bioinform*, 54.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Mach. Learn*, 73(2):133?153.

Geller, J. (1997). Challenge: How ijcai 1999 can prove the value of ai by using ai. *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*.

Goldsmith, J., & Sloan, R. (2007). The ai conference paper assignment problem. *In Proceedings of the AAAI Workshop on Preference Handling in AI*.

Grimmett, G., & Stirzaker, D. (1982). Probability and Random processes. *Oxford University Press*.

Gui, G., & Scott, P. (2007). Ranking reusability of software components using coupling metrics. *J. Syst. Softw*, 1450–1459.

Herlocker, J., Konstan, J., & Riedl, J. (2000). Explaining collaborative filtering recommendations. *In Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (CSCW)*.

Herlocker, J., Konstan, J., & Riedl, J. (2002). An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr*, 287–310.

Hettich, S., & Pazzani, M. (2006). Mining for proposal reviewers: Lessons learned at the national science foundation. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* .

Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., & Kusumoto, S. (2003). Component rank: Relative significance rank for software component search. *In Proceedings of the 25th International Conference on Software Engineering (ICSE)*.

Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., & Kusumoto, S. (2005). Ranking significance of software components based on use relations. *IEEE Trans. Software Eng*, 213–225.

Jin, R., & Si, L. (2004). A study of methods for normalizing user ratings in collaborative filtering. *In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Jin, R., Si, L., Zhai, C., & Callan, J. (2003). Collaborative filtering with decoupled models for preferences and ratings. *In Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM)*.

jun Zeng, H., cai He, Q., Chen, Z., ying Ma, W., & Ma, J. (2004). Learning to cluster web search results. *In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Kahng, M., Lee, S., & goo Lee, S. (2011). Ranking in context-aware recommender systems. *In Proceedings of the 20th International Conference on World Wide Web (WWW)*.

Kendall, M., & Gibbons, J. (1990). Rank Correlation Methods. *Edward Arnold, London, UK*.

Kleinberg, J. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, 604?632.

Klementiev, A., Roth, D., & Small, K. (2008). Unsupervised rank aggregation with distance-based models. *In Proceedings of the 25th International Conference on Machine Learning (ICML)*.

Kumar, R., & Vassilvitskii, S. (2010). Generalized distances between rankings. *In Proceedings of the 19th International World Wide Web Conference (WWW)*.

Lauw, H., Lim, E.-P., & Wang, K. (2006). Bias and controversy: Beyond the statistical deviation. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Lauw, H., Lim, E.-P., & Wang, K. (2007). Summarizing review scores of "unequal" reviewers. *In Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*.

Lauw, H., Lim, E.-P., & Wang, K. (2012). Quality and leniency in online collaborative rating systems. *ACM Trans. Web*, 4:1–4:27.

Lemire, D. (2005). Scale and translation invariant collaborative filtering systems. *Inf. Retr*, 129–150.

Liu, N., & Yang, Q. (2008). Eigenrank: A ranking-oriented approach to collaborative filtering. *In Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

Liu, N., Zhao, M., & Yang, Q. (2009). Probabilistic latent preference analysis for collaborative filtering. *In Proceedings of the 18th ACM conference on Information and Knowledge Management (CIKM)*.

Liu, Q., Yang, J., & Xu, Y. (2010). In-silico prediction of blood secretory human proteins using ranking algorithm. *BMC Bioinform*, 250.

Liu, Y.-T., Liu, T.-Y., Qin, T., Ma, Z.-M., & Li, H. (2007). Supervised rank aggregation. *In Proceedings of the 16th International World Wide Web Conference (WWW)*.

Mallows, C. (1957). Non-null ranking models. *Biometrika*, 44:114-130.

Mamoulis, N., Yiu, M., Cheng, K., & Cheung, D. (2007). Efficient top-k aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):1168.

Marden, J. (1995). Analyzing and Modeling Rank Data. *Chapman & Hall, New York*.

Marsh, H., & Roche, L. (1997). Making students' evalutions of teaching effectiveness effective. *American Psychologist*, 52(11):1187–1197.

Mimno, D., & Mccallum, A. (2007). Expertise modeling for matching papers with reviewers. *In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Ng, V. (2005). Machine learning for coreference resolution: From local classification to global ranking. *In Proceedings of the 43rd Annual Meeting of the Asssociation for Computational Linguistics (ACL)*.

Ng, V. (2005). Supervised ranking for pronoun resolution: Some recent improvements. *In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. *In Proceedings of the 7th International World Wide Web Conference (WWW)*.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. *In Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW)*.

Riggs, T., & Wilensky, R. (2001). An algorithm for automated rating of reviewers. *In Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*.

Robertson, S., Walker, S., Beaulieu, M., Gatford, M., & Payne, A. (1995). Okapi at trec-4. *In Proceedings of the 4th Text Retrieval Conference (TREC)*.

Saranli, A., & Demirekler, M. (2001). A statistical unified framework for rank-based multiple classifier decision combination. *Patt. Recog.*, 34(4):865–884.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). Application of dimensionality reduction in recommender systems: A case study. *In WebKDD Workshop at the ACM SIGKKD*.

Shen, J., Lin, Y., Xue, G.-R., Zhu, F.-D., & Yao, A.-G. (2006). Irfcf: Iterative rating filling collaborative filtering algorithm. *In Proceedings of the 8th Asia-Pacific Web Conference on Frontiers of WWW Research and Development (APWeb)*.

Shi, Y., Larson, M., & Hanjalic, A. (2010). List-wise learning to rank with matrix factorization for collaborative filtering. *In Proceedings of the 4th ACM conference on Recommender systems (RecSys)*.

Su, X., & Khoshgoftaar, T. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*.

Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H., & Wu, T. (2009). Rankclus: Integrating clustering with ranking for heterogeneous information network analysis. *In Proceedings of the 12th International Conference on Extending Database Technology (EDBT)*.

Sun, Y., Yu, Y., & Han, J. (2009). Ranking-based clustering of heterogeneous information networks with star network schema. *In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Traupman, J., & Wilensky, R. (2006). Collaborative quality filtering: Establishing consensus or recovering ground truth? *In Proceedings of the 6th International Conference on Knowledge Discovery on the Web: Advances in Web Mining and Web Usage Analysis (WebKDD)*.

Walpole, R., Myers, R., Myers, S., & Ye, K. (2002). Essentials of Probability & Statistics for Engineers & Scientists. *Prentice Hall, NJ*.

Wang, F., Chen, B., & Miao, Z. (2008). A survey on reviewer assignment problem. *In Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*.

Wang, K., & Su, M. (2002). Item selection by "hub-authority" profit ranking. *In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Weimer, M., Karatzoglou, A., Le, Q., & Smola, A. (2007). Cofirank: Maximum margin matrix factorization for collaborative ranking. *In Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*.