Preference Aggregation in Group Recommender Systems for Committee Decision-Making

Jacob Baskin, Google
Shriram Krishnamurthi, Brown University

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-09-07
June 2009

Preference Aggregation in Group Recommender Systems for Committee Decision-Making

Jacob Baskin
Google, Inc.
jbaskin@google.com

Shriram Krishnamurthi
Brown University
sk@cs.brown.edu

ABSTRACT

We present a preference aggregation algorithm designed for situations in which a limited number of users each review a small subset of a large (but finite) set of candidates. This algorithm aggregates scores by using users' relative preferences to search for a Kemeny-optimal ordering of items. We use variable-neighborhood local search, allowing the efficient discovery of high-quality consensus orderings, which facilitates effective categorization of candidates while remaining computationally feasible for large problem instances. This algorithm provides a significant increase in solution quality over existing systems when evaluated against real-world data, as well as when tested against benchmark problem instances. We further discuss potential applications of this algorithm in group recommender systems for a variety of scenarios, including program committees and faculty searches, and discuss a number of considerations regarding its use.

1. INTRODUCTION

In situations such as conference program committees and faculty searches, a group of people are tasked with reviewing a large set of candidates, and then collaboratively identifying which of these candidates are best. This problem is within the domain of *group recommender systems*. While most recommender systems use the preferences of a large group of people to extrapolate those of an individual, group recommender systems aggregates individuals' preferences to recommend the best items for the group as a whole.

Such systems present challenges that do not face recommender systems designed for individuals [11]. In particular, these challenges include the need to accurately and fairly aggregate users' individual preferences, and the difficulty of designing a user interface that facilitates collaborative decision making.

One of the simplest methods for aggregating user preferences is eliciting numerical ratings, and then assigning each item a score that is a function of all the individual scores it receives. While this is a viable technique for some systems [16], it fails within the context of committee decision-making, for a number of reasons:

1. It is not clear what a given numerical score "means"; in particular, different reviewers may have different perceptions of how good or bad an item should be to receive a particular score. Assigning textual descriptions to the scores explicitly does not fully solve this problem, as those descriptions themselves can be interpreted differently. Moreover, allowing only scores that correspond to descriptions may place overly restrictive limits on the user's ability to express preferences.

2. Some reviewers are universally negative or universally positive. When every user does not express a preference on every item, this tendency will result in items being considered more favorably solely because they happen to have been reviewed by a positive person, or more negatively if they happen to have been reviewed by a negative person.

3. Some reviewers may also, by chance or by inclination, review only high-quality items or only low-quality items. This situation is very hard to distinguish from that of universally positive or negative reviewers; if we assume reviewers who always give high scores are universally positive, we will penalize items that happen to be reviewed by those who have seen many good items.

Nonetheless, numerical scores are appealing from a user-interface perspective; users are accustomed to assigning numeric scores, or analogous indicators such as a number of "stars", as a means of providing preference information. But how can we glean more dependable information from these numbers?

We propose a group recommender system that considers only the *relative preferences* of each reviewer: which items they prefer to others. Regardless of how high or low the actual scores are, we take it as axiomatic that if reviewer A gives item 1 a higher score than item 2, she prefers 1 to 2. Thus, if we extract from each reviewer's score data an ordering of the items he reviewed, and find a way to aggregate these orderings, we can combine preferences elicited through standard numerical comparisons without encountering any of the above problems.

Contributions. This paper presents both a novel local-search algorithm for finding near-optimal consensus orderings and a strategy for using these orderings to assist groups tasked with reviewing large numbers of different items. We tested the local search algorithm against other heuristic algorithms for rank aggregation, both on existing problem sets and on random problems designed to be similar to the data encountered in committee decision-making scenarios. Our algorithm achieves much better solutions for large and hard problems than existing algorithms, most of which have been optimized for quickly finding solutions to smaller, more "well-behaved" instances; additionally, it solves most smaller problems to optimality. We are also optimistic that this algorithm can scale up to produce high-quality (though non-optimal) results on even larger data sets. We analyzed this recommender system using anonymized data from various real-world situations, such as academic conferences.

2. AGGREGATING ORDERINGS

In order to make use of reviewers' relative preferences, we must find a way to combine the disparate preferences of reviewers together. Aggregating orderings—combining the relative preferences of multiple people into a single "consensus ordering" in as fair of a way as possible—is the very same problem addressed by voting theory [1]. Accordingly, it has been much studied, beginning in the 18th century with Borda [6] and Condorcet [4]. More recently, Kemeny [13] proposed a rule whereby the consensus ranking should be chosen to minimize the number of *violations*: the number of cases where the consensus ordering ranks C_i as better than C_j, but some reviewer R_k prefers C_j to C_i.

The Kemeny rule has a number of desirable properties as a preference aggregation function [21, 23]. However, it has a number of less-desirable properties:

- Finding the optimal Kemeny ordering is NP-hard. Indeed, it is even NP-hard to approximate better than 1.36... [7, 12]. Furthermore, branch-and-bound methods to find optimal rankings cannot handle more than 80-100 items [3, 5]; as conferences, jobs, and graduate schools routinely have hundreds of applicants, and larger-scale recommender scenarios may have thousands or even millions of different items to rank, a larger-scale method is required.
- Even if we can find an optimal or near-optimal order, it will be of only limited use to the user. There may be multiple optimal orderings; presenting any particular one as the "correct" ordering of the items ignores a number of important issues. An item that has received few reviews, for instance, may have a highly variable place in the ordering.
- A consensus ordering is only a relative picture of the items' quality: we do not know whether the 15th-best item is "good" or not, only that it is better than the 16th-best, 17th-best, and so forth. Lastly, if items are scored on multiple criteria, there may be different orderings that must be integrated with each other.
- Any ranking, no matter how good, would not optimally serve the goal of facilitating committee decision-making: almost invariably, final decisions are produced within meetings, in which committee members consider some subset of the items as a group. Rather than providing a putative ordering of all the items, a recommender system should prune the total search space in order to limit the number of spurious items the committee must consider together.

3. CHOOSING AN AGGREGATION METHOD

Different group recommender systems use different methods of eliciting and aggregating preferences: MusicFX [16] asks for scores on a scale of -2 to 2, and computes each item's score based on a function of the individual scores it receives. *Travel Decision Forum* elicits scores from its users, and offers a number of different aggregation mechanisms, among them taking the median of all users' scores [11]. Others do not even conceive of user preferences as scores at all: Lorenzi et al. [15] concieve of user preferences as a set of constraints, and attempt to find recommendations that fit the constraints of all the users.

We choose a score-based method of eliciting preferences, but we then aggregate these scores based on their implied ranking of items. Our aggregation method–computing a Kemeny-optimal ranking–is ideal for committee decision-making, for a number of reasons.

First, unlike simple score-based schemes, our algorithm is robust in situations where not every user expresses a preference for every item, as we describe in the introduction.

Second, the goal in scenarios such as program committees and employee searches is, generally, not to find items that are acceptable to as many group members as possible, but to find a subset of items that have a high likelihood of containing the best possible item. Some of the best items may be the most contentious; it is important that conflict of this type not rule out these items, as it would in a constraint-based system. At the same time, it is important that we identify such conflict, as knowing which items are contentious ahead of time can facilitate the decision-making process. The Kemeny ranking gives us a good tool for identifying conflict, but does not rule out conflict-causing items. For more information on how we use the Kemeny ranking to identify and manage conflict, see our categorization algorithm in section 5.

Lastly, using a score-based method allows us to adapt our system to situations that users are already accustomed to. Most users expect that they will be assigning some sort of numerical or ordinal scores to items in committee decision-making situations; by accepting such scores as input, our system can co-exist with existing group decision-making processes that depend on scores or other such rankings, such as Identify the Champion [17], which we summarize in section 5.

One apparent flaw of the technique of deriving rank-orderings from scores is its introduction of "dependence" between users' scores. If users know that each score that they give will be analyzed in comparison to their other scores, this may make it more complicated for users to find the right score to give an item. The effect of assigning a given score to a given item

suddenly depends on all the other scores that were assigned by that same user. Scores, however, are never truly independent, regardless of how they are aggregated. In any situation where items will be compared with each other, the "value" of having a particular score depends on how many items have higher or lower scores. Moreover, in most cases, there is no reason to expect that the actual scores users ought to assign to reflect their true preferences are any different if the scores are dependent in this way.

An Axiomatic View

Pennock et al. discuss recommendation algorithms in the context of social choice theory, identifying a number of axioms that are desirable for recommender systems to possess [18]. These are:

1. Universal domain and minimal function: every item is ranked, and the system can produce an output for every possible set of input scores.
2. Unanimity: if every user prefers A to B, then A will be ranked above B.
3. Independence of irrelevant alternatives: No matter what users' utilities are for some item C, the rankings of any two items A and B do not change if we removes C from consideration.
4. Scale independence: if, for each user x, we multiply all of that user's scores by some constant α_x and add some constant β_x, the output rankings will not change.

Our system trivially satisfies the "universal domain and minimal function" axiom; furthermore, any system that considers only relative preferences will satisfy scale independence, since scaling a user's scores does not affect their relative preferences. This is an improvement over the weighted-average aggregation proposed by Pennock et al., which does not satisfy this axiom.

A Kemeny-optimal ordering, because it satisfies the extended Condorcet criterion [22], also satisfies the "unanimity" axiom when preferences are consistent: i.e., if, whenever a user prefers A to B and B to C, that user also prefers A to C. Deriving preferences from scores ensures this consistency.

However, our algorithm makes no attempt to satisfy the "independence of irrelevant alternatives" condition. In fact, "irrelevant" alternatives can often give us useful data in the face of incomplete information: if there is no user who compares some item A to another item B directly, we can use users' preferences with respect to all the other items under consideration to make an indirect determination of A's relative quality with respect to B. This is what allows us to generate recommendations in the face of sparse information. Thus, we consider this axiom too strong for our domain, since it precludes the possibility of obtaining much of the information necessary to integrate reviewers' preferences.

4. ALGORITHM FOR PREFERENCE AGGREGATION

Figure 1: Variable Neighborhood Search Algorithm

```
O ← findLocalMax(randomOrder())
bestOrder ← O
for i = 1 ... numIters do
    for k = 1 ... k_max do
        O ← diversify_k(O)
        O ← findLocalMax(O)
        if C(O) < C(bestOrder) then
            bestOrder ← O
            break
        else if C(O) > C(bestOrder) then
            O ← bestOrder
        end if
    end for
end for
```

In this section, we present a local search algorithm that quickly determines optimal or near-optimal consensus rankings according to the Kemeny criterion from sparse ranking data. In our categorization algorithm below, we use these rankings in concert with original scores to determine whether items are good or bad, as well as to identify conflict.

We can use reviewers' preference data to construct a weighted directed graph $G = (V, E)$, in which each vertex $v \in V$ represents an item being reviewed, and the edge from v_i to v_j has weight c_{ij} equal to the number of reviewers who prefer i to j. Given this graph, finding the Kemeny-optimal ordering becomes an instance of the Linear Ordering Problem (LOP), in which the goal is to find a total order $>_o$ that minimizes $L(o) = \sum_{i,j \,:\, i>_o j} c_{ij}$.

Another formulation of this problem is to find the order that maximizes $U(o) = \sum_{i,j \,:\, i>_o j} c_{ji}$. We choose to use the $L(o)$ formulation, as it closely corresponds to the "minimal-violations" description of the Kemeny criterion. While both formulations are equivalent, they do have implications when judging the performance of algorithms. See section 4.4 below for further discussion.

4.1 High-Level Algorithm

Our algorithm has the same structure as the variable neighborhood search of Garcia et al. [8], in figure 1 above. We begin with a random ordering, and proceeds to a local maximum. At every iteration, we then try to improve on that maximum by performing a series of successively larger "diversification" moves, followed by the finding of another local maximum. Each diversification move attempts to change the solution enough that the subsequent search for a local maximum will find a maximum that is distinct from bestOrder. If the local maximum represented by bestOrder is close to a better maximum, it should take only a small change in our solution to move us towards this higher point; however, if this does not work, we try larger moves. This is the essence of variable neighborhood search [8]. As soon as we reach a local maximum better than the one where we started, the series starts all over again, with the smallest diversification moves.

We improve on Garcia et al.'s variable neighborhood search

Figure 2: cascadeMoveUp procedure

```
procedure CASCADEMOVEUP(p)
    p_o ← p
    c_o ← 0
    for i = p − 1 ... 0 do
        c_i ←The cost of moving the vertex at p to i
        if c_i ≤ c_o then
            c_o ← c_i
            p_o ← i
        end if
    end for
    if p_o > 0 then
        cascadeMoveUp(p_o − 1)
        cascadeMoveUp(p_o)
    end if
end procedure
```

Figure 3: cascadeAll procedure

```
procedure CASCADEALL
    repeat
        c_old ← C(O)
        for i = N − 2 ... 0 do
            cascadeMoveDown(i)
        end for
        for i = 1 ... N − 1 do
            cascadeMoveUp(i)
        end for
    until C(O) = c_old
end procedure
```

algorithm in two places: we have a novel technique for finding local maxima, and we modify the diversification step to be more appropriate for sparse graphs. These techniques result in major improvements, especially on the particular problem instances encountered in rank aggregation.

4.2 Finding Local Maxima: "Cascade"

Most attempts to find good local maxima for the linear ordering problem have used a BESTFIT search, in which the algorithm examines each vertex, and inserts that vertex in the position that results in the most improvement of the objective function. When none of the vertices can be moved in a way that improves the objective function, the algorithm terminates. While this does not produce a very good heuristic for the linear ordering problem on its own [19], many algorithms have used it to good effect for finding improvements on solutions arrived at by other means. [10, 3].

Our algorithm improves on BESTFIT by performing the insertion moves in an order more likely to result in a high local maximum. To do this, we use CASCADEMOVEUP, shown in figure 2, and a similar CASCADEMOVEDOWN.

The intuition behind these moves is that the cost of switching the vertices in p_o and $p_o - 1$ (or $p_o + 1$ in the MoveDown case) must be greater than 0, or else $p_o - 1$ rather than p_o would be the optimal position for the vertex at p. However, it is possible that moving the vertex in position p_o farther *would* decrease the cost if the vertex at $p_o - 1$ were not "in the way". Thus, we first try and move the vertex in $p_o - 1$ as far as we can, and then try once again to move the vertex in position p_o.

We can combine these two moves into a heuristic for finding local optima, by performing each of them on all vertices in order of their position. This is the CASCADEALL procedure, described in figure 3.

In practice, we have found that this procedure works much better than using simple BESTFIT, particularly on sparse matrices such as those produced in rank aggregation. In particular, for real-world data, we have found that this move produces results 75% closer to optimal on average than BESTFIT. On the other hand, it also performs significantly more slowly, as the recursion results in each move examining the same vertices repeatedly. But if we remember which vertices are "stuck"—already unable to move further—and use this information to terminate the recursion early, we can achieve a substantial speedup: with this optimization, CASCADE takes only 50-75% longer than BESTFIT, as opposed to over 200% longer without it.

4.3 Diversification: Adapting to Sparse Problem Instances

In a typical conference, job search, or social Web site, most pairs of items have not been compared directly with each other. This means that the graph of reviewers' relative preferences will be very sparse. This poses a problem, because it results in there existing many moves that, despite changing the position of a vertex, do not substantially change the solution: if the only result of a move is that vertices with no connection to each other are re-arranged, then it is unlikely this move will result in the discovery of a new local optimum.

Garcia et al. use a diversification step diversify_k which performs k random moves in which a randomly-chosen vertex is placed in a randomly chosen position other than its own. We modify this move by identifying, for a vertex k, the vertices $pred(k)$ and $succ(k)$—the closest vertices ordered lower than k and higher than k, respectively, to which k is directly connected. Our diversification step uses only random moves that switch relative order of k and either $pred(k)$ or $succ(k)$.

4.4 Computational Experiments

In the following tables of algorithm results, each cell contains three numbers. The first, labeled "V", is the average value of the solutions produced by the algorithm on the given problem set, according to the $L(o)$ metric; for preference-aggregation problems, this is the total number of "violations" in the preference graph. The lower this value, the better the algorithm performed. The second number, labeled "D", is the average difference between the $L(o)$ value produced by this algorithm and the best value found by any algorithm. The third number, labeled "Best", is the number of times the algorithm produced the best result of any of the four. The algorithm that performed the best overall on a given problem set is printed in bold.

Figure 4: Results for Random Type 2

	Size 100	Size 150	Size 200
VNS	**V: 111569.36** **D: 0.00** **Best: 25**	V: 385614.64 D: 1.92 Best: 12	V: 928062.52 D: 8.00 Best: 8
VNS-D	O: 111569.92 D: 0.56 Best: 23	V: 385616.64 D: 3.92 Best: 8	V: 928064.28 D: 9.76 Best: 9
VNS-C	**V: 111569.36** **D: 0.00** **Best: 25**	V: 385614.56 D: 1.84 Best: 13	**V: 928060.52** **D: 6.00** **Best: 7**
VNS-CD	V: 111569.52 D: 0.16 Best: 24	**V: 385613.52** **D: 0.90** **Best: 18**	V: 928060.92 D: 6.40 Best: 6

Figure 5: Results for Random Rank Aggregation

	400 Dense	300 Dense	400 Sparse	300 Sparse
VNS	V: 2605.2 D: 14.2 Best: 1	V: 1589.6 D: 3.8 Best: 0	V: 539.5 D: 3.7 Best: 1	V: 446.5 D: 1.2 Best: 6
VNS-D	V: 2601.6 D: 10.6 Best: 1	V: 1593.2 D: 7.4 Best: 1	V: 541.1 D: 5.3 Best: 0	V: 448.2 D: 2.9 Best: 2
VNS-C	V: 2596.5 D: 5.5 Best: 4	V: 1587.0 D: 1.2 Best: 5	V: 537.0 D: 1.2 Best: 5	**V: 445.9** **D: 0.6** **Best: 7**
VNS-CD	**V: 2595.8** **D: 4.8** **Best: 4**	**V: 1586.9** **D: 1.1** **Best: 5**	**V: 536.4** **D: 0.6** **Best: 6**	V: 446.7 D: 1.4 Best: 5

Evaluating Solutions: L(o) vs. U(o). Most local-search algorithms' reports list the $U(o)$ values of the solutions discovered by the algorithms. Much like primal and dual linear programs, these two formulations are equivalent: any solution with a good $L(o)$ value will have a good $U(o)$ value, and vice-versa. However, which formulation is chosen does affect how the results of heuristic algorithms are judged. If $\frac{L(o)}{U(o)}$ is large, as it is in some of the instances found in the popular LOLIB problem set [20], then an algorithm that produce near-optimal solutions when judged using the $U(o)$ formulation will also appear near-optimal when judged by the $L(o)$ formulation. However, problems arising from rank aggregation tend to have much smaller values for $\frac{L(o)}{U(o)}$, as there tend to be a relatively small number of "violations" relative to "non-violations". Indeed, real-world preference data from reviewers in a faculty search shows $\frac{L(o)}{U(o)}$ values as low as 10% of those for LOLIB instances. This means that this same algorithm would appear to be much worse when judged using $L(o)$.

We evaluated our algorithm against the real-world data derived from a departmental faculty search with roughly 350 applicants and 25 reviewers, against the instances in LOLIB [20], against the "Random Type 2" instances generated by Campos et al. [2], and against a set of instances randomly generated to be similar to real-world rank aggregation instances. these instances were generated by simulating k reviewers

Figure 6: Results for Real-World Rank Aggregation

	Potential	Fit
VNS	V: 328.89 D: 3.6 Best: 1	V: 308.33 D: 4.8 Best: 0
VNS-D	V: 328.11 D: 3.3 Best: 1	V: 308.78 D: 4.4 Best: 1
VNS-C	**325.67** **D: 0.7** **Best: 5**	V: 306.89 D: 3.1 Best: 1
VNS-CD	V: 326.11 D: 1.0 Best: 4	**V: 305.56** **D: 2.0** **Best: 4**

each ranking p of n papers. Their rankings are each set initially to be identical, but are then each perturbed by m random moves. The rankings are then combined into an instance of the linear ordering problem. For each of $n = 300$ and $n = 400$ we generated both "sparse" instances ($k = n/10, p = 30, m = 15$) and "dense" instances ($k = n/5, p = 50, m = 20$). We generated 10 each of these 4 types, for 40 instances in total.

On each of these instances, we ran four variants of our algorithm:

- **VNS**: our algorithm without the CASCADE heuristic or the modifications to the diversification step; essentially identical to the algorithm in [8].
- **VNS-D**: our algorithm with the modifications to the diversification step.
- **VNS-C**: our algorithm with the CASCADE heuristic.
- **VNS-CD**: our algorithm with both the CASCADE heuristic and the modifications to the diversification step.

For each of these algorithms, we set $k_{\max} = 20$, while Garcia et al. set $k_{\max} = 10$; we found that using a high $k_{\max}$ improved solutions for rank-aggregation problems in particular. We ran each algorithm for the same amount of processor time, rather than setting maxIters to a particular value; this allowed us to determine whether the additional time taken by the CASCADE heuristic is put to better use than it would be by simply adding more iterations.

All algorithms were executed for 5 seconds of processor time on an Intel Pentium M processor running at a clock speed of 1.4GHz.

On LOLIB instances, each algorithm reached the optimal solution for every problem, with the exception of **VNS-CD**, which returned a solution 2 below optimal on `be75np`, a benchmark problem instance corresponding to the input/output matrix of the Belgian economy's national production in 1975. We do not believe there were any specific difficulties with this instance in particular; rather, we hypothesize that this was just a poor run caused by the effects of randomness.

On Random Type 2 instances, our modifications to the diversification step made no substantial differences, but the CASCADE heuristic improved the algorithm. The numbers below show the average scores of each type of algorithm on each set of instances, the average distance from the optimal solution, and the number of best solutions.

For the random rank-aggregation instances, the CASCADE heuristic consistently improved on BESTFIT. The modifications to the diversification step resulted in slight improvements; surprisingly, these improvements were greater for dense instances than for sparse instances. Additionally, the **VNS-D** algorithm was the worst of the four on three of the four sets of problems; it seems that the CASCADE heuristic and the diversification changes interact positively, but the diversification changes alone do not improve the solutions.

For the problems encountered from real-world faculty-search data, our algorithm provides a real improvement over "standard" VNS. For the "Fit" data set, which was sparser, VNS-CD performed best (and reached the best solution which much more regularity than the other algorithms), while for "Potential", both VNS-C and VNS-CD were roughly equal.

As can be seen in the results, the **VNS-CD** algorithm, while it appears to show gains over **VNS** for hard problems in general, succeeds especially in the domain of rank aggregation. In fact, this is the only domain in which the changes made to the diversification step are of any use; in other domains, our diversification step sometimes results in a worse heuristic. But in the particular area of Kemeny rank aggregation, both these modifications and the CASCADE heuristic are effective.

5. CATEGORIZATION ALGORITHM

While the **VNS-CD** algorithm yields a good ordering of the vertices, just finding a consensus ordering is not enough, as explained in section 2. In order to provide more useful, higher-quality information than a consensus ordering alone could give, we use a categorization algorithm incorporating both the original scores and the consensus ordering. This algorithm produces a a reliable and consistent picture of which items are good and which are bad, as well as identifying items that sparked reviewer conflict, taking into account relative preferences and beliefs about objective quality. Moreover, the algorithm acts over multiple consensus orderings, allowing our system to compare items with respect to multiple review criteria.

To produce a categorization using a single consensus ordering, we first attempt to move unconstrained vertices to better positions. This is accomplished by using the $succ(p)$ and $pred(p)$ functions defined in section 4.3. Since moving a vertex to any position in the order between its predecessor's position and its successor's would not affect the cost, all that our consensus ordering really tells us about a vertex's position is that it should be below its predecessor and above its successor. Thus, we create O_h, an order sorted by $succ(p)$ which we use to look for good vertices, and O_l, an order sorted by $pred(p)$, to use when looking for bad vertices.

The next step is anchoring the consensus ordering. To accomplish this, we return to the "raw" scores. Our goal is to figure out approximately which numerical score is equivalent to which position in the sorted order. Thus, we compute a moving average of scores—the average of the average scores within a given region in the ordering. Looking at vertices' average scores in isolation would result in the same biases we have set out to correct; on the other hand, we expect average scores to be predictive of quality most of the time. Thus, by taking a moving average, we can use the consensus ordering while still getting a rough idea of the score to which a given position in the ordering corresponds.

Using user-supplied cutoffs c_{good} and c_{bad}, we then find the "cutoff vertices" in O_h and O_l—the first vertex in O_h with a moving average score below c_{good}, and the first vertex in O_l with a moving average score above c_{bad}. Every vertex that is orderd before these vertices in O_h and O_l, respectively, are labelled "good" and "bad".

The next step is identifying vertices that are "in conflict". To accomplish this, we look at all "good" vertices $v \in G$, and calculate the percentage of preferences that compare v and some non-"good" vertex u in which u is ranked above v:

$$\text{conflict}(v) = \frac{\sum_{u \notin G} c_{uv}}{\sum_{u \notin G} c_{uv} + c_{vu}}$$

If this value exceeds the cutoff value c_{conflict}, v is labelled as "good but conflicted". A similar technique is used to identify "bad but conflicted" vertices.

To produce a consensus ordering for multiple review criteria, we combine the categories obtained by using the scores and consensus orderings for each criterion individually. For a vertex to be considered good overall, it must be categorized as good for each criterion; a similar rule applies for bad vertices. To be conisdered conflicted, however, a good (or bad) vertex need only be identified as conflicted on a single criterion.

Real-World Effectiveness

We evaluated our algorithm using data from a number of small conferences that used "Identify the Champion" [17] scoring. However, it is important to note that the number of papers ranked as "good" or "bad" was very dependent on the score cutoffs used; which cutoffs were effective varied by conference. Some cutoffs would produce zero "good" or zero "bad" papers for one conference, even though they produced reasonable breakdowns for others. This may be a result of the small range of scores offered by Identify the Champion, or it may be the case that deciding how many applicants to select as "good" and as "bad" using score-based cutoffs is not particularly effective.

To incorporate review information as fully as possible, we incorporated reviewers' expertise ratings into our algorithm. If reviewer r prefers paper a to b, and has experience e_a for a and e_b for b, we weight the reviewers' preference by $min(e_a, e_b)$. We assigned numerical values of 3, 4, and 5 to the three different experience levels specified in ItC [17]. The intention is not to discount reviews by non-experts, but to ensure that if there is a direct disagreement between an expert and a non-expert, the expert's opinion will override the non-expert's.

Identify the Champion (ItC) is a system that is designed to enable program committees to more efficiently determine which submissions to accept to a conference. It works by determining if any reviewer is prepared to "champion" a given submission—to argue strongly for its acceptance. Reviewers have four different ways to score a paper:

A: Good paper. I will champion it at the PC meeting.

B: OK paper, but I will not champion it.

C: Weak paper, though I will not fight strongly against it.

D: Serious problems. I will argue to reject this paper.

Ideally, the committee will quickly accept papers with "A"s but no "D"s, and quickly reject all papers without any "A"s or "B"s, and only devote extended discussion to papers with both "A"s and "D"s.
The authors' experience, however, suggests that while ItC is good at deciding which papers to reject out-of-hand, few papers are accepted quickly. "A" scores are not very prevalent, meaning that often, committee time is spent on papers with only "B" and "C" ratings, for which ItC provides no guidance.

Our algorithm, running on commodity hardware, produced optimal results in under five seconds for all conferences; thus, this algorithm's performance is suitable for use in real-time or near-real-time applications in scenarios such as these.

We found that our algorithm was slightly worse at identifying which papers would be accepted than ItC. While we never identified an accepted paper as "bad", we occasionally identified as "good" papers that would go on to be rejected. However, our algorithm labeled more papers as "good" or "bad" than were given scores of A and D; for papers to which no reviewer gave either of these ratings, our algorithm correctly identified which papers would go on to be accepted 80% of the time, which provided an improvement over the minimal information otherwise available.

It is worthwhile noting, furthermore, that these conferences do not represent the target environment for our recommender system. All four conferences used for evaluation were small (between 10 and 36 submissions), and all four used the scoring system designed for ItC; our technique is designed for conferences with many more submissions, in which accpeting all A-rated papers may not be an option; it is also intended to be used with a scoring system with more granularity.

6. OTHER RELATED WORK

Cook et al. have previously examined the problem of creating a consensus ranking from those of individual reviewers; a branch-and-bound algorithm is proposed that results in very fast optimal solutions for low-noise instances with up to 60 elements [5]. By using a heuristic local-search algorithm, we sacrifice the ability to obtain an optimal solution for increased scalability; our algorithm is capable of handling much larger problems in a reasonable amount of time. We also address the problem of unreliable consensus rankings, allowing our system to be more robust and to provide more dependable information. Additionally, by returning a categorization rather than an ordering of the input elements, our system supports its users in their decision-making process, rather than simply attempting to return an "optimal" answer. Lastly, the procedure designed by Cook et al. is meant to be used in concert with their allocation scheme for peer reviews, which is not always practicable, especially in domains such as faculty and graduate-school applications where reviewers must be given the opportunity to rate the applicants about whom they have personal knowledge.

Another approach to this problem was demonstrated by Hasan et al. [9] in the domain of trust recommendation. This approach replaces each score with its percentile—instead of revealing a score of 8/10, say, one would learn that a given reviewer ranked an item above 75% of the others she reviewed. While this approach is much computationally simpler, ours solves a number of problems that would plague this approach if applied to conferences, searches or similar situations. For instance, Hasan et al. assume that a person's "medium" score is reflected by the median of their observed scores. If a reviewer rates mostly good or mostly bad items, this may not be the case; as mentioned in section 4, we have encountered such behavior in real situations, with "counterpoint" reviewers. Additionally, this approach may obscure useful information that can be uncovered by rank aggregation; for instance, if we know that A is better than B and B is better than C, our algorithm will always (in the absence of conflicting information) place A above C, while this is not necessarily the case when using percentiles.

Other local search algorithms for the Linear Ordering Problem have been proposed, using metaheuristics such as tabu search [14], scatter search [2], and simulated annealing [19]; Huang and Lim have also developed a genetic algorithm for the Linear Ordering Problem [10]. However, most of these algorithms were designed for instances such as those encountered in the Triangulation Problem for Input-Output Matrices in economics; indeed, the popular LOLIB library of sample instances for the linear ordering problem are derived entirely from this source, and tend to be both smaller (60 elements or fewer) and denser than the problems encountered in our domain. Our algorithm is designed specifically for large, sparse problem instances, resulting in much improved behavior for rank aggregation in particular; in addition, the introduction of the Cascade method of finding local optima results in improvements over existing algorithms on all problems.

7. CONCLUSION AND FUTURE WORK

This paper has presented an improved local search algorithm for aggregating user preference data based on variable neighborhood search. We have discussed the Cascade procedure, which generates significantly better local optima than the BestFit heuristic used in the past. This procedure, along with a strategy for more effectively diversifying solutions in "sparser" problem instances, yields major improvements when applied to real-world data sets.

We also discussed a recommender system that uses the consensus ordering produced by this local search procedure to help find which of the items under review are good and which are bad, as well as to identify conflict between reviewers over particular items. One major advantage of this system over others is its ability to identify conflict. For small conferences, while this algorithm does not improve the ability of "Identify the Champion"' [17] to reliably identify papers that will be accepted, it does give more useful information about papers in the middle of the pack. Combining our ranking approach with "Identify the Champion" may result in a very effective system, and would be an interesting direction for future research on this topic.

Much more empirical data is required to give a good assessment of the efficacy of this recommender system in comparison to less-sophistocated methods; this data will be created as the algorithm is implemented and used in various recommender systems. It may be possible to design effective systems for this task that do not rely on solving the NP-hard Linear Ordering Problem to find a consensus ordering. Additionally, finding better ways of reducing the variability of consensus rankings—perhaps identifying and extricating disconnected components of the preference graph, for instance—could result in algorithms that are based on Kemeny rank aggregation being less susceptible to strange patterns of reviews such as those discussed in section 4 above. In general, using relative preference data to provide feedback that is richer than a "mere" consensus ranking may prove to be a fruitful direction for future group recommender systems and may be the key to providing accurate, stable and unambiguous ways of separating good choices from bad.

8. REFERENCES

[1] Kenneth J. Arrow. *Social Choice and Individual Values*. Yale University Press, second edition, September 1970.

[2] Vicente Campos, Manuel Laguna, and Rafael Martí. Scatter search for the linear ordering problem. pages 331–340, 1999.

[3] Irène Charon and Olivier Hudry. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097–2116, 2006.

[4] Condorcet, marquis de. *Essai sur l'application de l'analyse á la probabilité des décisions rendue á la pluralité des voix*. Paris: Imprimerie royale, 1785.

[5] Wade D. Cook, Boaz Golany, Michael Penn, and Tal Raviv. Creating a consensus ranking of proposals from reviewers' partial ordinal rankings. *Computers & Operations Research*, 34(4):954–965, 2007.

[6] Jean-Charles de Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[7] Irit Dinur and Shmuel Safra. The importance of being biased. In *STOC '02: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 33–42, New York, NY, USA, 2002. ACM.

[8] Carlos G. Garcia, Dionisio Pérez-Brito, Vicente Campos, and Rafael Martí. Variable neighborhood search for the linear ordering problem. *Computers and Operations Research*, 33(12):3549–3565, 2006.

[9] Omar Hasan, Lionel Brunie, Jean-Marc Pierson, and Elisa Betino. Elimination of subjectivity from trust recommendation. Technical Report CSD TR-08-008, Purdue University, West Lafayette, IN 47907, 2008.

[10] Gaofeng Huang and Andrew Lim. Designing a hybrid genetic algorithm for the linear ordering problem. In *Genetic and Evolutionary Computation Conference*, pages 1053–1064, 2003.

[11] Anthony Jameson. More than the sum of its members: challenges for group recommender systems. In *AVI '04: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 48–54, New York, NY, USA, 2004. ACM.

[12] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, NY, 1972. Plenum Press.

[13] J. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.

[14] Manuel Laguna, Rafael Marti, and Vicente Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26(12):1217–1230, 1999.

[15] Fabia Lorenzi, Fernando Santos, Jr. Paulo R. Ferreira, and Ana L. Bazzan. Optimizing preferences within groups: A case study on travel recommendation. In *SBIA '08: Proceedings of the 19th Brazilian Symposium on Artificial Intelligence*, pages 103–112, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] Joseph F. McCarthy and Theodore D. Anagnost. MusicFX: an arbiter of group preferences for computer supported collaborative workouts. In *CSCW '98: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 363–372, New York, NY, USA, 1998. ACM.

[17] Oscar Nierstrasz. Identify the champion. In N. Harrison, B. Foote, and H. Rohnert, editors, *Pattern Languages of Program Design*, volume 4, pages 539–556. Addison Wesley, 2000.

[18] David M. Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 729–734. AAAI Press / The MIT Press, 2000.

[19] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8, 2003.

[20] Gerhard Reinelt. LOLIB, 1997. `http://www.iwr.uni-heidelberg.de/groups/comopt/soft/LOLIB/`.

[21] Donald Saari and Fabrice Valognes. Geometry, voting, and paradoxes. *Mathematics Magazine*, 71(4):243–259, 1998.

[22] Michel Truchon. An extension of the Condorcet criterion and Kemeny orders. Cahiers de recherche 9813, Université Laval - Département d'économique, 1998.

[23] H.P. Young and A. Levenglick. A consistent extension of Condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.