Preference Aggregation in Group Recommender Systems for Committee Decision-Making

Jacob P. 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, and then uses this ordering to identify good and bad items, as well as those that are the subject of reviewer conflict. The algorithm uses variable-neighborhood local search, allowing the efficient discovery of high-quality consensus orderings while remaining computationally feasible. It provides a significant increase in solution quality over existing systems. We discuss potential applications of this algorithm in group recommender systems for a variety of scenarios, including program committees and faculty searches.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Performance

Keywords

Preference aggregation, Local search algorithms

1. INTRODUCTION

In situations such as conference program committees and faculty searches, a group of people is tasked with reviewing a large set of candidates, and then collaboratively identifying which of these are suitable. This problem is within the domain of *group recommender systems*, which aggregate 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 [7]; in particular, they must both accurately and fairly aggregate users' individual preferences.

RecSys'09 , October 22–25, 2009, New York, USA.

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 [11], it fails within the context of committee decision-making, for a number of reasons. First, different reviewers may have different perceptions of how good or bad an item should be to receive a particular score, obscuring each score's "meaning". Second, some reviewers are universally negative or universally positive. When every user does not express a preference on every item, items will be considered differently on the basis of who happens to have reviewed them. Third, 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. 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. If a given reviewer gives a higher score to item A than to item B, we take it as axiomatic that that reviewer prefers A to B. Our system uses the preferences thus exposed to extract a partial ordering of items from each reviewer's score data, and aggregates these orderings, in order to combine preferences elicited through standard numerical comparisons without encountering any of the above problems.

To accomplish this, we use both a novel local-search algorithm for aggregating individuals' orderings into a single consensus ordering, and a strategy for using these orderings to assist groups tasked with reviewing large numbers of different items. Our algorithm produces better results, on average, than any existing heuristics, and it solves most smaller problems to optimality. We analyzed this recommender system using anonymized data from various real-world situations, such as academic conferences.

An extended version of this paper, with details of the algorithms, their implementation, and evaluation, is available at `www.cs.brown.edu/research/pubs/techreports/reports/CS-09-07.html`. Our generated data sets are available at `www.cs.brown.edu/research/plt/dl/recsys2009/`.

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]. Kemeny [8] proposed a rule whereby the consensus ordering should be chosen to minimize the number of *violations*: the number of triples (C_i, C_j, R_k) such that reviewer R_k prefers item C_i to C_j, but the consensus ordering ranks C_j above C_i.

The Kemeny rule has a number of desirable properties as a preference aggregation function [15]. However, it has a number of less-desirable properties:

1. Finding the optimal Kemeny ordering is NP-hard. Exact methods to find optimal rankings cannot handle more than 80-100 items [3, 4]. Recommender systems may have to deal with hundreds, thousands, or even millions of different items.

2. There are mutliple Kemeny-optimal orderings, and any given item's position may vary widely between them. Presenting any particular one as the "correct" ordering of the items ignores this important issue.

3. A consensus ordering presents 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.

4. Almost invariably, final decisions are produced within meetings, in which committee members consider some subset of the items as a group. Thus, an ordering is not the most useful output to provide. Instead, a recommender system should prune the total search space in order to limit the number of spurious items the committee must consider together.

In order to address these issues, we use a local search algorithm to quickly provide a near-optimal answer rather than an exact solution; furthermore, we do not use this consensus ordering directly, but rather use it as input to our categorization system, described in section 4, which aims to provide simple, useful, and dependable information based on this ordering.

Using relative preferences allows us to provide robust information from scores. While scores are a common method of eliciting preferences in group recommender systems [7, 11], they are not universal. Lorenzi et al. [10], for instance, concieve of user preferences as a set of constraints, and attempt to find recommendations that fit the constraints of all the users. However, such systems tend to eliminate items that would cause conflict, which is not the goal in scenarios such as program committees and employee searches. We should *highlight* conflict-causing items rather than penalizing or eliminating them. Our ranking provides the additional benefit of identifying conflict without ruling out conflict-causing items.

3. ALGORITHM FOR PREFERENCE AGGREGATION

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 $L(o) = \sum_{i,j\ :\ i>_o j} c_{ij}$.

3.1 High-Level Algorithm

Our algorithm is a variable-neighborhood search algorithm, with the same structure as the algorithm described by Garcia et al. [5]. 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 the previously best-known order. If the best local maximum we have found so far 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 [5]. 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 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.

3.2 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 repeatedly examines each vertex, and inserts that vertex in the position that results in the most improvement of the objective function. This continues until no moves can be found that improve the result.

We improve upon BestFit with cascadeMoveUp (Figure 1) and the analogous cascadeMoveDown, which we developed to exploit the structure found in sparse matrices. We combine these two moves into a heuristic for finding local optima: first, we go from vertex $N-2$ to vertex 1, performing cascadeMoveDown, and then from 2 to $N-1$ performing cascadeMoveUp. We repeat these two steps until we achieve no further improvement.

In practice, we have found that this procedure works much better than using simple BestFit. On the other hand, it also performs significantly more slowly. 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.

3.3 Diversification for Sparse Instances

Garcia et al. use a diversification step diversify$_k$ which performs k random moves in which a randomly-chosen vertex is

Figure 1: cascadeMoveUp procedure

procedure CASCADEMOVEUP(p)
    $p_o \leftarrow p$
    $c_o \leftarrow 0$
    for $i = p - 1 \ldots 0$ **do**
        $c_i \leftarrow$The cost of moving the vertex at p to i
        if $c_i \leq c_o$ **then**
            $c_o \leftarrow c_i$
            $p_o \leftarrow i$
        end if
    end for
    if $p_o > 0$ **then**
        cascadeMoveUp($p_o - 1$)
        cascadeMoveUp(p_o)
    end if
end procedure

placed in a randomly chosen position other than its own. k begins at 1, and each time the algorithm fails to find a new local maximum, k increases until it reaches k_{max}. We modify this move by identifying, for a vertex v, the vertices $pred(v)$ and $succ(v)$—the closest vertices ordered lower than v and higher than v, respectively, to which v is directly connected. Our diversification step uses only random moves that switch relative order of v and either $pred(v)$ or $succ(v)$, which improves its effectiveness on the sparse matrices prevalent in preference aggregation.

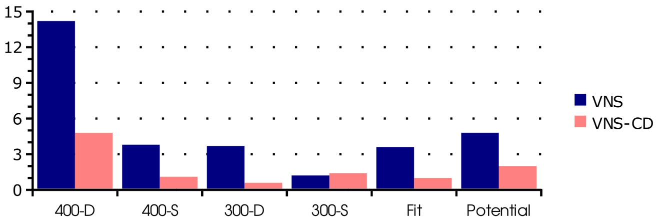
3.4 Computational Experiments

We evaluated our algorithm against real-world data obtained from a departmental faculty search with roughly 350 applicants and 25 reviewers, against the instances in LOLIB [14], 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 each ranking p of n papers. Their rankings are each set initially to be identical, but are then each perturbed by m random moves. 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. The dense instances are called **300-D** and **400-D** and the sparse instances **300-S** and **400-S**. There are two data sets from the faculty search: one is called **Potential**, corresponding to reviewers' ratings on a "Faculty Potential" scale; the other **Fit**, corresponding to their ratings on a "Goodness of Fit" scale.

On each of these instances, we ran two different algorithms. The first, **VNS**, comprised the variable-neighborhood search algorithm with none of our modifications, as decribed in [5]. The second, **VNS-CD**: was our variable neighborhood search algorithm with the CASCADE heuristic and the modifications to the diversification step.

For these algorithms, we set $k_{\max} = 20$, while Garcia et al. set $k_{\max} = 10$; we found that using a high $k_{\max}$, allowing our search to diverge farther from previous local maxima, improved solutions for rank-aggregation problems in particular. We ran each algorithm for the same amount of processor time, rather than performing a set number of iterations; 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.

Figure 2: Average Distance from Best Result



All algorithms were executed for 5 seconds of processor time on an Intel Pentium M processor at 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 one instance, `be75np`. We hypothesize that this was just a poor run caused by the effects of randomness.

For the random rank-aggregation instances, our algorithm produced significant improvements in most cases; however, it was marginally worse on the "300 sparse" set. For the problems encountered from real-world faculty-search data, our algorithm provides a real improvement over "standard" VNS. Thus, our **VNS-CD** algorithm is an improvement over the state of the art in the domain of rank aggregation.

4. CATEGORIZATION ALGORITHM

While the **VNS-CD** algorithm yields a good ordering of the vertices, just finding a consensus ordering is not enough. To improve the quality and utility of our information, we use a categorization algorithm that incorporates both the original scores and the consensus ordering.

We first attempt to reduce variability by gaining information about unconstrained vertices. This is accomplished by using the $succ(p)$ and $pred(p)$ functions defined in section 3.3. Since moving a vertex to any position in the order between its predecessor's position and its successor's would not affect the cost, a vertex's position within that range is arbitrary. 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 the initial scores. We computing a moving average of items' average scores, in order to figure out approximately which numerical score is equivalent to which position in the sorted order without re-introducing the same biases we have set out to correct. Starting with the highest vertex in O_h, we label all vertices as "good" until the moving average of scores first falls below some user-supplied cutoff values c_{good}; similarly, we label vertices as "bad" in O_l until the moving average of scores rises above some c_{bad}.

The final 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.

Real-World Effectiveness

We evaluated our algorithm using data from a number of small conferences that used the "Identify the Champion" (ItC) system for scoring [12]. In this system, reviewers score papers from A to D, where A signifies that the reviewer will "champion" the paper by arguing for its acceptance, and D signifies that the reviewer will argue for the paper's rejection. B and C are intermediate values signifying views not strong enough for the reviewer to actively argue for or against.

We found that our algorithm (which produced optimal consensus orderings within five seconds on commodity hardware for all conferences) 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 definitive recommendations by ItC, which is designed so that papers with all As but no Ds should be accepted, and all Ds but no As should be rejected. For papers with no As or Ds, 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. Moreover, it is worth noting that these small conferences, each having between 10 and 36 submissions, do not represent the target environment for our recommender system.

5. 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 [4]. By using a heuristic local-search algorithm, we sacrifice the ability to obtain an optimal solution for increased scalability. Our categorization system provides information about conflict, while addressing the problem of unreliable consensus rankings, allowing us to provide more complete and dependable information. 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 practicable in all domains.

Other local search algorithms for the Linear Ordering Problem have been proposed, using various metaheuristics [2, 6, 9, 13]. 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.

6. FUTURE WORK

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. 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 much more robust against conflicting reviews. 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.

Acknowledgements

We thank Meinolf Sellman, Claire Mathieu, Warren Schudy, and David Laidlaw for their helpful comments on our system, Arjun Guha for his help testing and supporting it, and Pascal Van Hentenryck for his assistance optimizing the local search algorithm. This work is partially supported by the NSF. Baskin's work was conducted at Brown University.

7. REFERENCES

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

[2] V. Campos, M. Laguna, and R. Martí. Scatter search for the linear ordering problem. *New Ideas in Optimization*, 1999.

[3] I. Charon and O. Hudry. A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15), 2006.

[4] W. D. Cook, B. Golany, M. Penn, and T. Raviv. Creating a consensus ranking of proposals from reviewers' partial ordinal rankings. *Computers & Operations Research*, 34(4), 2007.

[5] C. G. Garcia, D. Pérez-Brito, V. Campos, and R. Martí. Variable neighborhood search for the linear ordering problem. *Computers and Operations Research*, 33(12), 2006.

[6] G. Huang and A. Lim. Designing a hybrid genetic algorithm for the linear ordering problem. In *Genetic and Evolutionary Computation Conference*, 2003.

[7] A. Jameson. More than the sum of its members: challenges for group recommender systems. In *Advanced Visual Interfaces*, 2004.

[8] J. Kemeny. Mathematics without numbers. *Daedalus*, 88, 1959.

[9] M. Laguna, R. Marti, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26(12), 1999.

[10] F. Lorenzi, F. Santos, J. Paulo R. Ferreira, and A. L. Bazzan. Optimizing preferences within groups: A case study on travel recommendation. In *Brazilian Symposium on Artificial Intelligence*, 2008.

[11] J. F. McCarthy and T. D. Anagnost. MusicFX: an arbiter of group preferences for computer supported collaborative workouts. In *Computer Supported Cooperative Work*, 1998.

[12] O. Nierstrasz. Identify the champion. *Pattern Languages of Program Design*, 4, 2000.

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

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

[15] D. Saari and F. Valognes. Geometry, voting, and paradoxes. *Mathematics Magazine*, 71(4), 1998.