Improved Bounds for Computing Kemeny Rankings[*]

Vincent Conitzer
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
conitzer@cs.cmu.edu

Andrew Davenport
Mathematical Sciences Department
IBM T.J. Watson Research Center
Yorktown Heights, New York, 10598
davenport@us.ibm.com

Jayant Kalagnanam
Mathematical Sciences Department
IBM T.J. Watson Research Center
Yorktown Heights, New York, 10598
jayant@us.ibm.com

Abstract

Voting (or *rank aggregation*) is a general method for aggregating the preferences of multiple agents. One voting rule of particular interest is the *Kemeny rule*, which minimizes the number of cases where the final ranking disagrees with a vote on the order of two alternatives. Unfortunately, Kemeny rankings are NP-hard to compute. Recent work on computing Kemeny rankings has focused on producing good bounds to use in search-based methods. In this paper, we extend on this work by providing various improved bounding techniques. Some of these are based on cycles in the pairwise majority graph, others are based on linear programs. We completely characterize the relative strength of all of these bounds and provide some experimental results.

Introduction

A *voting* (or *rank aggregation*) rule takes as input multiple rankings of (or *votes* over) a fixed set of alternatives (or *candidates*), and aggregates them into a single concensus ranking of the candidates. Rank aggregation is a key problem for preference aggregation in multiagent settings, where agents must make joint decisions on joint plans, allocations of resources, *etc.*, in spite of conflicting preferences over the possible decisions. However, there are many other applications as well; for example, different search engines may rank pages differently, and we may wish to aggregate these rankings into a single ranking. Other applications include collaborative filtering [20] and planning among automated agents [15; 16]. Recent work in artificial intelligence and related areas has studied the complexity of executing voting rules [18; 5; 12; 10; 1]; the complexity of manipulating elections [7; 14; 13]; eliciting the votes efficiently [6]; and adapting voting theory to the setting where the candidates vote over each other by linking to each other (as in the context of the World Wide Web) [3; 2].

Many different voting rules have been proposed. Probably the best known rule is the *plurality* rule, which simply ranks candidates by how often they are ranked first in the votes. Of course, this ignores a lot of the information. The *Kemeny rule* [19] is another voting rule. The Kemeny rule is defined as follows: it produces a ranking that maximizes the number of *pairwise agreements* with the votes, where we have a pairwise agreement whenever the ranking agrees with one of the votes on which of a pair of candidates is ranked higher.

The Kemeny rule has an important interpretation as a maximum likelihood estimator of the "correct" ranking. Condorcet, an early social choice theorist, modeled elections as follows: there is a correct ranking of the candidates, but every voter only has a noisy perception of this correct ranking. (We note that such a model makes sense especially in settings such as the one in which the "voters" are search engines that rank pages.) Specifically, for every pair of candidates, any voter ranks the better candidate higher with probability $p > 1/2$, independently.[1] Given this noise model, the problem of finding the maximum likelihood estimate of the correct outcome, given the votes, is well-defined. Condorcet solved this problem for the cases of 2 and 3 candidates [11]. Over two centuries later, Young observed that the Kemeny rule is in fact the solution to Condorcet's problem for arbitrary numbers of candidates [22]. Because of this, the Kemeny rule is sometimes also referred to as the Kemeny-Young rule. Recently, Conitzer and Sandholm showed that some, but not all, of the other well-known voting rules can also be interpreted as maximum likelihood estimators, under different (more complex) noise models [8].

Unfortunately, it is NP-hard to compute Kemeny rankings [4], even with only four votes [12]. The problem of computing Kemeny rankings is receiving increasing attention. Some work has focused on finding rankings with a Kemeny score that is *approximately* optimal, in polynomial time [1]. Although this approach has produced some interesting results, it also suffers from a few fundamental drawbacks. An approximation algorithm for a voting rule is, in effect, a different voting rule; and in real-world elections, voters may feel deceived if a different voting rule is used than the one that was promised to them. To illustrate this

[*]This material is based on work done while the first author was visiting IBM's T.J. Watson Research Center in the context of an IBM Ph.D. Fellowship.

[1]Of course, the rankings of pairs of candidates cannot actually be completely independent, because if a is preferred to b, and b to c, then a must be preferred to c. Nevertheless, all rankings receive some probability under this model, which is all that is necessary for the maximum likelihood approach.

point, the well-known Borda voting rule actually provides a good approximation of the optimal Kemeny score [9]. Moreover, some of the suggested approximation schemes do not seem very sensible as a voting rule. For example, choosing one of the votes at random as the ranking is a good approximation [1]. Yet another issue is that voters who are aware of which approximation scheme is being used may have an incentive to vote differently than they would if an optimal ranking were computed. (Of course, by the Gibbard-Satterthwaite impossibility result [17; 21], no reasonable voting rule is strategy-proof.)

Because of these issues, in this paper, we focus on computing *optimal* Kemeny rankings. Specifically, we will look at search-based methods. A key factor that determines the runtime of such methods is the availability of good bounds on the Kemeny score, and providing such bounds is the topic of this paper. Recently, Davenport and Kalagnanam provided a bounding technique for computing Kemeny rankings based on finding edge-disjoint cycles in a graph whose vertices are the candidates [10]. In this paper, we extend upon this technique. Specifically, we show how one can obtain bounds from *overlapping* cycles. We also provide various bounds based on linear programs, and show how these bounds relate to the cycle-based bounds.

Definitions

The Kemeny rule

For any two candidates a and b, given a ranking r and a vote v, let $\delta_{a,b}(r, v) = 1$ if r and v agree on the relative ranking of a and b (they either both rank a higher, or both rank b higher), and 0 if they disagree. Let the *agreement* of a ranking r with a vote v be given by $\sum_{a,b} \delta_{a,b}(r, v)$, the total number of pairwise agreements. A *Kemeny ranking* r maximizes the sum of the agreements with the votes, $\sum_{v} \sum_{a,b} \delta_{a,b}(r, v)$.

Reinterpretation of the Kemeny rule

We will now reinterpret the Kemeny rule as a *minimization* problem on a weighted directed graph. (Hence, the lower bounds that we provide in the paper correspond to upper bounds on the Kemeny score.) The vertices on this graph correspond to the candidates. The edges and their weights are given as follows. For every pair of candidates a, b, there is a number of votes $h_{(a,b)}$ preferring a to b, and a number of votes $h_{(b,a)}$ preferring b to a. If $h_{(a,b)} > h_{(b,a)}$, then draw a directed edge from a to b with weight $w_{(a,b)} = h_{(a,b)} - h_{(b,a)}$; conversely, if $h_{(b,a)} > h_{(a,b)}$, then draw a directed edge from b to a with weight $w_{(b,a)} = h_{(b,a)} - h_{(a,b)}$. If $h_{(a,b)} = h_{(b,a)}$, draw no edge at all.

Given an edge from a to b, if the final ranking ranks a above b, this corresponds to $h_{(a,b)}$ pairwise agreements; whereas if the final ranking ranks b above a, this corresponds to only $h_{(b,a)}$ pairwise agreements, that is, $h_{(a,b)} - h_{(b,a)} = w_{(a,b)}$ fewer agreements. In other words, every time that the final ranking does not agree with the direction of an edge, we lose a number of pairwise agreements that is equal to the weight of that edge. Therefore, the Kemeny problem can be restated as follows: find the ranking which minimizes the total weight of the edges that it disagrees with.

Lower bounds based on cycles in the graph

In this section, we describe various classes of lower bounds that are based on cycles in the graph described above. We start with the class of bounds given by Davenport and Kalagnanam. In this class of bounds, we take a set of edge-disjoint cycles, and add together the cycles' minimum weights (*i.e.* the weight of each cycle's lightest edge.)

Theorem 1 *For any set of edge-disjoint cycles C, $\sum_{c \in C} \min_{e \in c} w_e$ is a valid lower bound.*

Proof: For each cycle, the final ranking must be inconsistent with at least one of the edges in the cycle, and thereby lose the weight on that edge. ■

The lower bounding technique in Theorem 1 has the property that once we use an edge e in a cycle c, we cannot use any of its weight w_e again in another cycle. This is wasteful because c's lightest edge e' may have a much smaller weight $w_{e'}$, and we will only be able to increase the bound by $w_{e'}$. It turns out that we do not need to use all of e's weight, but rather only $w_{e'}$ of it: for, if we set apart $w_{e'}$ weight from each edge in the cycle, we know that in the final ranking, for at least one edge in the cycle, the $w_{e'}$ weight that we set apart from it will be lost. Hence we retain $w_e - w_{e'}$ of e's weight to use in other cycles through e.

This leads to the following class of lower bounds:

Theorem 2 *For any sequence of cycles $c_1, c_2, \ldots, c_l$, let $\delta(c, e)$ be an indicator variable that is 1 if $e \in c$, and 0 otherwise; and for $1 \leq i \leq l$, let $v_i = \min_{e \in c_i}(w_e - \sum_{j=1}^{i-1} \delta(c_j, e) v_j)$. Then, $\sum_{i=1}^{l} v_i$ is a valid lower bound.*

This class of bounds is no weaker than the previous class:

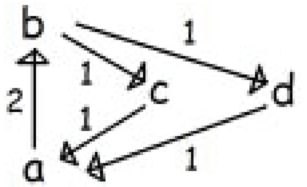
Theorem 3 *For any bound derived using Theorem 1, the same bound can be derived using Theorem 2.*

Proof: Given the set of l cycles C used according to Theorem 1, give these cycles an arbitrary order $c_1, c_2, \ldots, c_l$. Because the cycles are edge-disjoint, we have, for any c_i, $e \in c_i$, and $j < i$, that $\delta(c_j, e) = 0$. Therefore, $v_i = \min_{e \in c_i}(w_e - \sum_{j=1}^{i-1} \delta(c_j, e) v_j) = \min_{e \in c_i} w_e$. Hence this sequence of cycles used according to Theorem 2 gives the same bound. ■

In fact, the new class is strictly stronger:

Theorem 4 *On some instances, strictly stronger bounds can be derived using Theorem 2 than using Theorem 1.*

Proof: Consider the following instance:



Every cycle has a minimum weight of 1; once one cycle is removed entirely, no more cycles remain, and hence Theorem 1 cannot give us a better lower bound than 1. However, using Theorem 2, if we remove weight 1 from cycle (a, b, c) first, then we can still remove weight 1 from cycle (a, b, d), improving the lower bound to 2 (matching the cost of the optimal solution). ■

Rather than always choosing the minimum weight of any edge in the cycle as the amount to take from each edge in the cycle, we could also choose a smaller amount, without affecting the argument for why this constitutes a lower bound. This leads to the following class of lower bounds:

Theorem 5 *For any set of cycles C and function $v : C \to \mathbb{R}^+$, let $\delta(c, e)$ be an indicator variable that is 1 if $e \in c$, and 0 otherwise. Then, if for every $e \in E$, $\sum_{c \in C} \delta(c, e)v(c) \leq w_e$, then $\sum_{c \in C} v(c)$ is a valid lower bound.*

Again, this class of bounds is no weaker than the previous class:

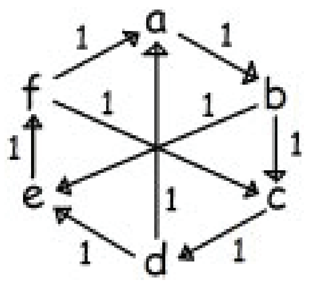
Theorem 6 *For any bound derived using Theorem 2, the same bound can be derived using Theorem 5.*

Proof: Given the sequence of cycles $c_1, c_2, \ldots, c_l$ used according to Theorem 2, let C be the set of all of these cycles, and let $v(c_i) = v_i = \min_{e \in c_i}(w_e - \sum_{j=1}^{i-1} \delta(c_j, e)v_j)$. ■

It is perhaps not immediately clear that this slightly generalized technique can give us better bounds, but it turns out that it can:

Theorem 7 *On some instances, strictly stronger bounds can be derived using Theorem 5 than using Theorem 2.*

Proof: Consider the following instance:



Because the weight on all edges is 1, removing the minimum weight from a cycle according to Theorem 2 will remove the entire cycle. That is, Theorem 2 can do no better than Theorem 1 on this instance. It is easy to verify that removing any cycle in this graph makes the graph acyclic; hence the best lower bound that can be obtained using Theorem 2 is 1. However, using Theorem 5, we can subtract weight 0.5 from each of the cycles $(a, b, c, d), (a, b, e, f), (c, d, e, f)$ to obtain a lower bound of 1.5, which can be rounded up to 2 since we know the solution cost must be integral. (This matches the cost of the optimal solution.) ■

In the next section, we will compare this section's bounds to bounds based on linear programs.

Lower bounds based on linear programs

In this section, we study bounds that are based on linear programs, and we compare them theoretically to the bounds based on cycles described in the previous section. In fact, the first linear program that we present corresponds to finding the optimal bound that can be obtained using Theorem 5. To find this optimal lower bound, we can use the following linear program (in which C is the set of all cycles in the graph):

Linear Program 1
maximize $\sum_{c \in C} y_c$
subject to for all $e \in E$, $\sum_{c \ni e} y_c \leq w_e$

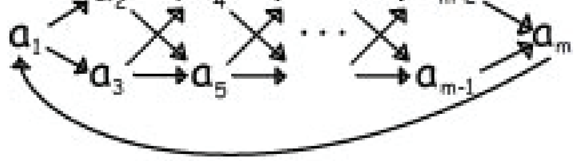
Let us consider the dual of this program, which is the following:

Linear Program 2
minimize $\sum_{e \in E} w_e x_e$
subject to for all $c \in C$, $\sum_{e \in c} x_e \geq 1$

This is in fact a linear program relaxation of an integer program formulation of the *minimum-edge feedback set* problem, which asks for the smallest set of edges to be removed from a given directed graph to make it acyclic. By the strong duality property of linear programs, its solution value must be the same as that of Linear Program 1. Thus, we obtain the following result:

Theorem 8 *The strongest lower bound that can be obtained using Theorem 5 is equal to the optimal solution to Linear Program 2.*

Unfortunately, Linear Programs 1 and 2 can have exponential size, because a directed graph can have exponentially many cycles. This is illustrated by the following example with $2^{(m-2)/2}$ cycles:



To remedy this, we now present a linear program that has polynomial size. This program is similar to the previous program, but it has a variable for *every* ordered pair of vertices, in contrast to the previous program which only has variables for the edges of the graph. On the other hand, it only has constraints for cycles of length 3.

Linear Program 3
minimize $\sum_{e \in E} w_e x_e$
subject to
for all distinct $a, b \in V$, $x_{(a,b)} + x_{(b,a)} = 1$
for all distinct $a, b, c \in V$, $x_{(a,b)} + x_{(b,c)} + x_{(c,a)} \geq 1$

If we add to this program the constraints that the variables $x_{(a,b)}$ must take integral values, then this in fact gives a mixed-integer program formulation for the Kemeny problem, in which the variable $x_{(a,b)}$ is set to 1 if a is ranked lower than b, and to 0 otherwise. The first set of constraints ensures that exactly one of the two candidates is ranked lower, and the second set of constraints enforces transitivity.

The objectives of Linear Programs 2 and 3 are the same; the spaces of feasible solutions, however, are not. First, Linear Program 3 also has variables for pairs of candidates with no edge between them, so if we wish to compare the feasible spaces of the linear programs we should restrict our attention to the variables that the programs have in common. More significantly, in Linear Program 2, we can set (for example) $x_e = 1$ for all the edges in a 3-cycle (a, b, c), whereas in Linear Program 3 this would not be feasible, because it would imply $x_{(a,c)} + x_{(c,b)} + x_{(b,a)} = 0$. Conversely, it turns out that the constraints in Linear Program 3 are (strictly) stronger:

Theorem 9 *Any feasible solution to Linear Program 3 is also feasible for Linear Program 2 (disregarding the variables that do not occur in the latter).*

Proof: Every constraint for a cycle of length 3 is clearly met (even for cycles that are not in C). We now prove that constraints for longer cycles are also met by induction on the length of the cycle. For a cycle $(v_1, v_2, \ldots, v_k)$ of length k, Linear Program 3 must satisfy $x_{(v_{k-1},v_k)} + x_{(v_k,v_1)} + x_{(v_1,v_{k-1})} \geq 1$ Moreover, by the induction assumption we must also have $x_{(v_1,v_2)} + x_{(v_2,v_3)} + \ldots + x_{(v_{k-2},v_{k-1})} + x_{(v_{k-1},v_1)} \geq 1$ Adding up these two inequalities, and using the fact that $x_{(v_1,v_{k-1})} + x_{(v_{k-1},v_1)} = 1$, we have $x_{(v_1,v_2)} + x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)} \geq 1$ as required. ■

It follows that the optimal solution value for Linear Program 3 is always at least as great as that for Linear Program 2. In fact, it turns out that the optimal solution values of both linear programs always coincide.

Theorem 10 *The optimal solution value for Linear Programs 2 and 3 is always identical.*

Proof: By the observations above, all that remains to show is that the optimal solution value for Linear Program 2 is never smaller than that for Linear Program 3. To show this, we transform an optimal solution to Linear Program 2 into a solution to Linear Program 3 without increasing its value. To do so, we must show how to set the additional variables in the latter program (we will keep the shared variables the same). For every variable $x_{(a,b)}$ in the former program, we must set $x_{(b,a)} = 1 - x_{(a,b)}$. We first show that doing so does not violate any cycle constraints (even for cycles of size larger than 3). Consider a constraint of the form $x_{(v_1,v_2)} + x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)} \geq 1$ and suppose that all of the variables $x_{(v_i,v_{i+1(\bmod k)})}$ in it have had their value set by the above, either because $(v_i, v_{i+1(\bmod k)}) \in E$ or because $(v_{i+1(\bmod k)}, v_i) \in E$. In the latter case ($(v_{i+1(\bmod k)}, v_i) \in E$), $x_{(v_i,v_{i+1(\bmod k)})} = 1 - x_{(v_{i+1(\bmod k)},v_i)}$. If $x_{(v_{i+1(\bmod k)},v_i)} = 0$, the constraint is immediately satisfied. Otherwise, there must be some cycle $(v_{i+1(\bmod k)}, v_i, w_{i,1}, w_{i,2}, \ldots, w_{i,l})$ in the original graph such that $x_{(v_{i+1(\bmod k)},v_i)} + x_{(v_i,w_{i,1})} + \ldots + x_{(w_{i,l},v_{i+1(\bmod k)})} = 1$ (for otherwise we could have reduced the value of $x_{(v_{i+1(\bmod k)},v_i)}$ in the solution to Linear Program 2 without violating any constraints, which, because all weights on edges are positive, would have improved the solution). Hence, we have $x_{(v_i,v_{i+1(\bmod k)})} = 1 - x_{(v_{i+1(\bmod k)},v_i)} = x_{(v_i,w_{i,1})} + \ldots + x_{(w_{i,l},v_{i+1(\bmod k)})}$. We can substitute this into the original constraint $x_{(v_1,v_2)} + x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)} \geq 1$ for every $x_{(v_i,v_{i+1(\bmod k)})}$ with $(v_{i+1(\bmod k)}, v_i) \in E$, at which point the constraint corresponds to a cycle in the original graph, and must therefore hold.

Next, we show that we can assign values to the additional variables, one at a time, so that no constraints are violated. Consider a pair of variables $x_{(v_1,v_2)}, x_{(v_2,v_1)}$ such that $(v_1, v_2) \notin E$ and $(v_2, v_1) \notin E$. For all constraints of the form $x_{(v_1,v_2)} + x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)} \geq 1$ for which all variables except for $x_{(v_1,v_2)}$ have been set already, consider the one that is furthest from being satisfied, that is, the one with the smallest quantity $x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)}$. Then, set $x_{(v_1,v_2)} = 1 - (x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)})$. (If there are no such constraints, simply set $x_{(v_1,v_2)} = 0$.) Now, we must set $x_{(v_2,v_1)} = 1 - x_{(v_1,v_2)} = x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)}$. All that remains to be proven is that this does not violate any constraints that include $x_{(v_2,v_1)}$, that is, constraints of the form $x_{(v_2,v_1)} + x_{(v_1,w_1)} + x_{(w_1,w_2)} + \ldots + x_{(w_l,v_2)} \geq 1$ for which all variables except for $x_{(v_2,v_1)}$ have been set already. But if we substitute $x_{(v_2,v_3)} + \ldots + x_{(v_k,v_1)}$ for $x_{(v_2,v_1)}$, this becomes a constraint for which all variables have already been set and which is therefore satisfied. ■

Denoting by *opt* the best lower bound that can be obtained using a particular family of bounds, or the optimal solution to a linear program, we have *opt*(Theorem 1) $\leq$ *opt*(Theorem 2) $\leq$ *opt*(Theorem 5) $=$ *opt*(Linear Program 1) $=$ *opt*(Linear Program 2) $=$ *opt*(Linear Program 3), where all of the inequalities are strict for some examples.

Experimental results

We performed an evaluation to compare the linear programming lower bound (Linear Program 3, or LP3) to the edge-disjoint 3-cycle lower bound [10] on randomly generated rank aggregation problems. For each problem, we first generated a total order representing a consensus ordering of m alternatives. We then generated the voting preferences, where each one of n voters agrees with the consensus ordering regarding the ranking of every pair of alternatives with some consensus probability p. We generated problems with 20, 30 and 40 alternatives and 5 voters. Each voter ranks all alternatives with no ties. We vary the consensus probability from 0.5 (no consensus at all) to 1.0 (full consensus with the consensus ordering). We generated 100 problems at each data point. We computed the edge-disjoint 3-cycle lower bound and the linear programming (LP3) lower bound for each problem. We also computed the optimal Kemeny distance for each problem, by formulating the problem as an integer programming problem using an integer formulation of the model LP3. We used CPLEX 9.1 to solve the linear and integer programming problems.[2] All experiments were performed on a 3GHz Pentium 4 machine. We present results showing the mean deviation of the edge-disjoint 3-cycle lower bound from the optimal Kemeny distance in Figure 1, the mean deviation of the linear programming lower bound in Figure 2, and the CPU times used to compute these bounds in Figures 3 and 4.

For most of the problems studied, the linear programming lower bound was equal to the optimal Kemeny distance. The linear programming lower bound was usually within 0.4% of the optimal Kemeny distance. The edge-disjoint 3-cycle lower bound performed relatively poorly in contrast. The CPU time required by the linear programming formulation appears to not scale as well as that for the edge-disjoint 3-cycle formulation. This is not surprising: for the linear programming lower bound, the size of the formulation is cubic in the number of alternatives in the problem, whereas the edge-disjoint 3-cycle formulation has size linear in the number of alternatives.

We also compared the CPU time used to find optimal Kemeny rankings by the branch-and-bound procedure of Davenport and Kalagnanam [10] to the CPU time used by CPLEX to solve the integer programming formulation derived from LP3. We present a comparison for problems with 25 alternatives and 5 voters in Table 1. At each consensus probability we solved 25 problems with a 300-second CPU time cutoff. We were able to solve all the problems using the integer programming approach. The branch-and-bound procedure was unable to solve many of the problems at low consensus probabilities.

[2]For problems with 40 alternatives we were unable to solve all problems to optimality using CPLEX within a reasonable time at consensus probabilities less than 0.58. We omit results for these data points.

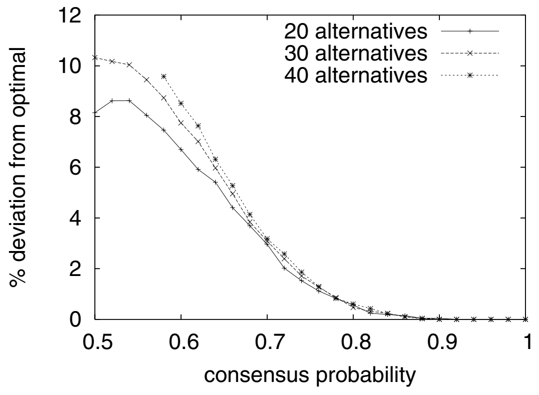


Figure 1: Mean deviation of edge-disjoint 3-cycle lower bound from optimal Kemeny distance for problems with 5 voters

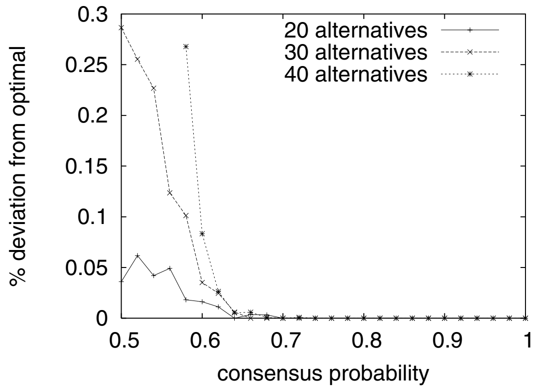


Figure 2: Mean deviation of linear programming relaxation (LP3) lower bound from optimal Kemeny distance for problems with 5 voters

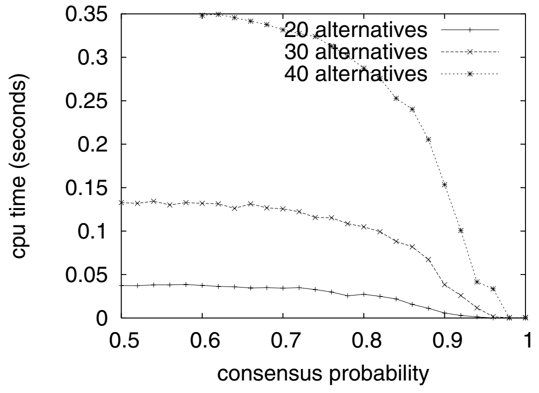


Figure 3: CPU time to compute edge-disjoint 3-cycle lower bound for problems with 5 voters

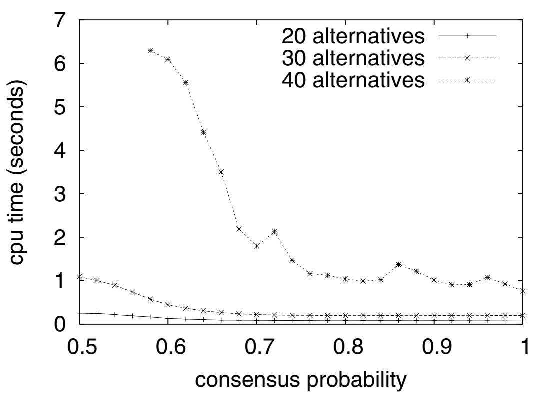


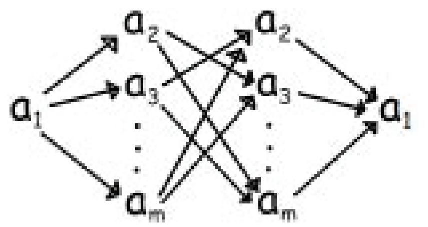
Figure 4: CPU time to compute linear programming relaxation (LP3) lower bound for problems with 5 voters

consensus probability	% solved B&B	mean CPU B&B (sec)	% solved IP	mean CPU IP (sec)
0.50	0	-	100	3.47
0.55	0	-	100	1.76
0.60	0	-	100	0.19
0.65	8	226	100	0.10
0.70	80	99.6	100	0.09
0.75	100	15.8	100	0.09
0.80	100	2.08	100	0.09
0.85	100	0.37	100	0.09
0.90	100	0.07	100	0.09
0.95	100	0.02	100	0.09
1.00	100	0.01	100	0.09

Table 1: A comparison of the CPU time required to solve to optimality Kemeny ranking problems with 25 alternatives and 5 voters. The branch-and-bound procedure of Davenport and Kalagnanam [10] (B&B) is compared to CPLEX solving the integer programming formulation corresponding to LP3 (IP).

Greedy cycle-based lower bounds

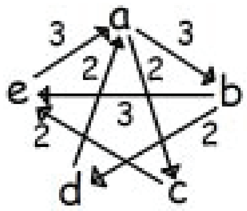
In spite of the fact that the lower bound obtained by solving one of the linear programs will always be at least as good as any of the cycle-based lower bounds, solving these linear programs will require more time at each search node than greedy methods for finding cycles in the graph. Because of this, search methods based on greedy techniques for finding cycles may in fact be faster. The greediest possible method is to find any cycle, remove its minimum weight from all of its edges, and repeat. However, longer cycles require us to subtract the weight from more edges without a corresponding improvement in the bound, hence it makes more sense to look for shorter cycles—specifically, cycles of length 3—first. Davenport and Kalagnanam give an algorithm for computing a good set of 3-cycles, but, as explained before, their method does not make use of overlapping cycles. We will give a somewhat similar method for finding 3-cycles that does make use of overlapping cycles. (In fact, being able to use overlapping cycles simplifies the method considerably.) In this method, we find weights to place on all the 3-cycles that run through a fixed vertex a_1 (in the sense of Theorem 5), in a way that maximizes the sum of these weights. Then we remove these weights from the edges, repeat for the next vertex a_2, *etc.* To find the weights for (say) a_1 we solve a MaxFlow instance over the following graph:



The capacity on each edge is the weight on that same edge in the pairwise majority graph if it exists, and 0 otherwise. Any path from the source to the sink corresponds to a 3-cycle through a_1, hence we can subtract the weight running over a path from the edges in that path. This procedure will never remove more than an edge's weight, because the edge capacities in the MaxFlow instance are equal to the weights in the pairwise majority graph (and an edge in the pairwise majority graph occurs at most once in the MaxFlow instance).

It is possible that, when there are no more 3-cycles remaining, there are still larger cycles remaining. Such cycles can still be exploited with a greedier method (the MaxFlow method described above does not work for cycles of length greater than 3, because in that case edges will occur more than once in the MaxFlow instance).

While this method will, in general, not produce optimal lower bounds, it can still avoid pitfalls of even greedier techniques. Consider, for example, the instance defined by the graph below.



(One Kemeny ranking for this instance is $b \succ c \succ d \succ e \succ a$, with a cost of 5.) An extremely greedy algorithm may first subtract weight 3 from the cycle (a, b, e) (the heaviest cycle), after which no cycles remain. However, if the MaxFlow approach described above starts by trying to maximize the weight on cycles through a, it will discover that placing weight 2 on (a, c, e), weight 1 on (a, b, e), and weight 2 on (a, b, d) gives a total lower bound of 5 (which is the cost of the optimal solution). (By contrast, if it starts on b, there are multiple ways of maximizing the weight on cycles through b, including placing weight 3 on (a, b, e) but also including placing weight 1 on (a, b, e), and weight 2 on (a, b, d), as in the optimal lower bound described above.)

Conclusions

Voting (or rank aggregation) is a key method for aggregating the preferences of multiple agents. One voting rule of particular interest is the *Kemeny rule*, which maximizes the number of pairwise agreements between the final ranking and the votes, and has an important interpretation as a maximum likelihood estimator. Unfortunately, Kemeny rankings are NP-hard to compute. This has resulted in the development of search-based algorithms for computing Kemeny rankings. A key factor in the runtime of such algorithms is the availability of strong admissible bounds.

In this paper, we first provided two new bounding techniques based on cycles in the pairwise majority graph. Both of these techniques are generalizations of Davenport and Kalagnanam's earlier edge-disjoint cycle bounding technique. In the first technique, we find a cycle, take the minimum weight of any of its edges and subtract it from all of its edges, and repeat. In the second (more general) technique, we subtract any nonnegative amount of weight from the edges of the cycle (as long as this does not produce an edge with negative weight). The first new technique is always at least as good as, and in some cases better than, the edge-disjoint cycle bounding technique; the second new technique is always at least as good as, and in some cases better than, the first new technique.

We then studied bounds based on linear programs. The first linear program computes the optimal bound that can be found using the second new cycle-based technique; its dual is in fact a linear program relaxation of an integer program formulation of the minimum-edge feedback set problem. Unfortunately, both of these programs require the enumeration of all the cycles in the graph, and there can be exponentially many cycles. To remedy this, we introduced another, polynomial-sized linear program that considered only triplets of candidates (rather than cycles of arbitrary length). We showed that the optimal solution values of all of these linear programs coincide.

Experimental results showed that the linear program bounds are significantly tighter than the edge-disjoint cycle bounds (and that running CPLEX on the integer programming formulation corresponding to one of these linear programs significantly outperforms the branch-and-bound procedure of Davenport and Kalagnanam [10] for computing Kemeny rankings). However, the linear program bounds take longer to compute. Thus, we introduced a greedy approach for computing bounds according to the second new cycle-based technique. This greedy approach maximizes the weight subtracted from 3-cycles involving a single candidate, and then continues on to the next candidate. Future research includes evaluating this approach experimentally.

References

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *STOC*, 2005.

[2] Alon Altman and Moshe Tennenholtz. On the axiomatic foundations of ranking systems. *IJCAI*, 2005.

[3] Alon Altman and Moshe Tennenholtz. Ranking systems: The PageRank axioms. *ACM-EC*, 2005.

[4] John Bartholdi, III, Craig Tovey, and Michael Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[5] William Cohen, Robert Schapire, and Yoram Singer. Learning to order things. *JAIR*, 10:213–270, 1999.

[6] Vincent Conitzer and Tuomas Sandholm. Vote elicitation: Complexity and strategy-proofness. *AAAI*, pages 392–397, 2002.

[7] Vincent Conitzer and Tuomas Sandholm. Universal voting protocol tweaks to make manipulation hard. *IJCAI*, pages 781–788, 2003.

[8] Vincent Conitzer and Tuomas Sandholm. Common voting rules as maximum likelihood estimators. *UAI*, pages 145–152, 2005.

[9] Don Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *SODA*, 2006.

[10] Andrew Davenport and Jayant Kalagnanam. A computational study of the Kemeny rule for preference aggregation. *AAAI*, pages 697–702, USA, 2004.

[11] Marie Jean Antoine Nicolas de Caritat (Marquis de Condorcet). Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. 1785. Paris: L'Imprimerie Royale.

[12] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. *WWW*, pages 613–622, 2001.

[13] Edith Elkind and Helger Lipmaa. Hybrid voting protocols and hardness of manipulation. *ISAAC*, 2005.

[14] Edith Elkind and Helger Lipmaa. Small coalitions cannot manipulate voting. *FC*, 2005.

[15] Eithan Ephrati and Jeffrey S Rosenschein. The Clarke tax as a consensus mechanism among automated agents. *AAAI*, pages 173–178, 1991.

[16] Eithan Ephrati and Jeffrey S Rosenschein. Multi-agent planning as a dynamic search for social consensus. *IJCAI*, pages 423–429, 1993.

[17] Allan Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[18] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *JACM*, 44(6):806–825, 1997.

[19] John Kemeny. Mathematics without numbers. In *Daedalus*, volume 88, pages 571–591. 1959.

[20] David M Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. *AAAI*, pages 729–734, 2000.

[21] Mark Satterthwaite. Strategy-proofness and Arrow's conditions: existence and correspondence theorems for voting procedures and social welfare functions. *J. Econ. Theory*, 10:187–217, 1975.

[22] Peyton Young. Optimal voting rules. *J. Econ. Persp.*, 9(1):51–64, 1995.