Improved Parameterized Algorithms for the Kemeny Aggregation Problem

Narges Simjour

June 8, 2009

Abstract

We give improvements over fixed parameter tractable (FPT) algorithms to solve the Kemeny aggregation problem, where the task is to summarize a multi-set of preference lists, called votes, over a set of alternatives, called candidates, into a single preference list that has the minimum total τ-distance from the votes. The τ-distance between two preference lists is the number of pairs of candidates that are ordered differently in the two lists. We study the problem for preference lists that are total orders. We develop algorithms of running times $O^*(1.403^{k_t})$, $O^*(5.823^{k_t/m}) \leq O^*(5.823^{k_{avg}})$ and $O^*(4.829^{k_{max}})$ for the problem, ignoring the polynomial factors in the O^* notation, where k_t is the optimum total τ-distance, m is the number of votes, and k_{avg} (resp, k_{max}) is the average (resp, maximum) over pairwise τ-distances of votes. Our algorithms improve the best previously known running times of $O^*(1.53^{k_t})$ and $O^*(16^{k_{avg}}) \leq O^*(16^{k_{max}})$ [4, 5], which also implies an $O^*(16^{4k_t/m})$ running time. We also show how to enumerate all optimal solutions in $O^*(36^{k_t/m}) \leq O^*(36^{k_{avg}})$ time.

1 Introduction

Preference lists are typical elements of psychology questionnaires and social science surveys. In many cases, we wish to combine the gathered preference lists into a single list that reflects the opinion of the surveyed group as much as possible. The Kemeny aggregation problem, introduced by Kemeny in 1959, is a famous abstract form of this problem [9]. Given a set of m total orders, called *votes*, over a set of n alternatives, called *candidates*, the Kemeny-optimal aggregation problem asks for a total order over candidates, called an *optimal aggregation*, that minimizes the sum of τ-distances from the votes, where the τ-distance between total orders π_1, π_2 is the number of pairs of candidates that are ordered differently in the two total orders.

Bartholdi et al. [2] proved that the problem is NP-hard. Later, Dwork et al. [8] showed that the problem remains NP-hard for constant even m's as small as $m = 4$. They used Kemeny's formalization [9] in their search for an effective spam filtering method that combined the results of multiple search engines. Dwork et al.'s article [8] initiated a series of papers studying algorithmic aspects of the Kemeny aggregation problem. The problem was shown to have an $O(n^{2.5} + mn^2)$ 2-approximation [8]. Ailon et al. developed randomized approximation algorithms of ratios 11/7 and 4/3 [1]. Later, Kenyon-Mathieu and Schudy developed a PTAS for the feedback arc set problem for special weighted tournaments, which solves the Kemeny aggregation problem as a special case [10]. Despite being a theoretical breakthrough, this algorithm could not be used in practise. Recently, in an attempt to develop practical approximation algorithms, Williamson and van Zuylen derived a deterministic 8/5-approximation algorithm for the problem [13]. The reader is referred to a survey by Charon and Hudry [6] for a detailed list of results.

Computational experiments of Davenport and Kalagnanam [7] suggest that the Kemeny aggregation problem might be easier to solve when an optimal aggregation is close to the input votes. In this direction, Betzler et al. [4] parameterized the problem with the sum of the τ-distances of the optimal aggregation from the input votes, denoted by k_t, and the maximum pairwise τ-distances of the input votes, denoted by k_{max}. They developed $O^*(1.53^{k_t})$ and $O^*((3k_{max} + 1)!)$ time algorithms, giving the first FPT algorithms for the Kemeny aggregation problem [4]. Later, they parameterized the problem by the average pairwise τ-distances of the input votes, denoted by k_{avg}, and developed an algorithm that ran in time $O^*(16^{k_{avg}}) \leq O^*(16^{k_{max}})$ [3, 5].

Our results. We develop parameterized algorithms of running times $O^*(1.403^{k_t})$, $O^*(5.823^{k_t/m}) \leq O^*(5.823^{k_{avg}})$, and $O^*(4.829^{k_{max}})$ for the problem, improving the previous best running times of $O^*(1.53^{k_t})$ [4] and $O^*(16^{k_{avg}}) \leq O^*(16^{k_{max}})$ [5]. We are the first to parameterized the problem in terms of k_t/m, although, by $k_{avg} \leq 4k_t/m$, any FPT algorithm in terms of k_{avg} implies an FPT algorithm in terms of k_t/m. We also give an algorithm to enumerate all optimal solutions in $O^*(36^{k_t/m})$ time. It is worth mentioning that k_t/m is smaller than k_{avg} and k_{max}, and therefore, parameterizing the problem with k_t/m, instead of k_{avg} or k_{max}, leads to potentially tighter analysis of FPT algorithms.

Figure 1 summarizes the running times proved in this paper and the best previous running times, in terms of the three parameters k_t, k_{avg} and k_{max}.

We fix pertinent notation in Section 2 and explain the parameterized algorithms in Section 3.

	Our Results			Previous Running Times		
m	k_t	k_{avg}	k_{max}	k_t	k_{avg}	k_{max}
3	1.403^{k_t}	$1.968^{k_{avg}}$	$1.968^{k_{max}}$			
4	1.403^{k_t}	$2.760^{k_{avg}}$	$2.760^{k_{max}}$			
5	1.342^{k_t}	$3.241^{k_{avg}}$	$3.241^{k_{max}}$			
6	1.342^{k_t}	$4.348^{k_{avg}}$	$4.348^{k_{max}}$			
...	...	...	...			
m	$(2.415^{1/\lceil m/2 \rceil})^{k_t}$	$5.833^{k_{avg}}$	$4.829^{k_{max}}$	1.53^{k_t} [4]	$16^{k_{avg}}$[5]	$16^{k_{max}}$ [5]

Figure 1: A summary of the running times proved in this paper and the best previous running times. Only the exponential terms are listed.

2 Preliminaries

We use U to denote the set of candidates. A *binary relation* on U is a subset of $U \times U$. A binary relation R is irreflexive if no (x, x), $x \in U$, is in R. A binary relation R is *asymmetric* if $(x, y) \in R$, $x, y \in U$ and $x \neq y$, implies $(y, x) \notin R$. In this article, we only work with irreflexive asymmetric binary relations. We may use $x <_R y$ to denote $(x, y) \in R$, and describe it as R orders x before y, .

A binary relation R is called *complete* if for any $x, y \in U$, $x \neq y$, either $(x, y) \in R$ or $(y, x) \in R$. A binary relation R is *transitive* if $(w, x) \in R$ and $(x, y) \in R$ imply $(w, y) \in R$. A *total order* is an irreflexive asymmetric binary relation that is complete and transitive. We use $\mathcal{T}_U$ to denote the set of total orders on U. We use R^+ for a transitive binary relation R to denote the transitive closure of R.

For any set $R \subseteq U \times U$, we use $rev(R)$ to denote $\{(b, a) : (a, b) \in R\}$; we may abuse the notation a little bit and use $rev((a, b))$ instead of $rev(\{(a, b)\})$. We say that $R_1 \subseteq U \times U$ is *consistent* with $R_2 \subseteq U \times U$ if $R_1 \cap rev(R_2) = \emptyset$.

Definition 1. *The* τ-distance *between* $\pi_1, \pi_2 \in \mathcal{T}_U$*, denoted by* $\tau(\pi_1, \pi_2)$*, is the cardinality of* $\pi_1 - \pi_2$*. For a multi-set* $\mathcal{I}$ *over* $\mathcal{T}_U$*,* $\tau(\pi_1, \mathcal{I})$ *is defined as the sum of* $\tau(\pi_1, \pi_2)$ *over all total orders* π_1 *in* $\mathcal{I}$*.*

An *optimal aggregation* of a multi-set $\mathcal{I}$ on $\mathcal{T}_U$ is a total order $\sigma \in \mathcal{T}_U$ that minimizes $\tau(\sigma, \mathcal{I})$. We use $OPT(\mathcal{I})$ to denote the set of all optimal aggregations. The *Kemeny aggregation problem* is the problem of finding an optimal aggregation for any given multi-set $\mathcal{I}$ on $\mathcal{T}_U$. For the case $|\mathcal{I}| = 1$ or

2, any $\sigma \in \mathcal{I}$ is an optimal aggregation [8]. Therefore, we are only interested in input instances that include more than two total orders.

We let $unanimity(\mathcal{I})$ denote the binary relation $\bigcap_{\pi \in \mathcal{I}} \pi$.

Observation 1. *[11] For any $\sigma \in OPT(\mathcal{I})$, $unanimity(\mathcal{I}) \subseteq \sigma$.*

Therefore, the Kemeny aggregation problem reduces to determining the order of dirty pairs, defined below:

Definition 2. *The set of* dirty pairs *of $\mathcal{I}$, denoted by $dirty(\mathcal{I})$, is $\{\{a, b\} : (a, b) \in \bigcup_{\pi \in \mathcal{I}}(\pi - unanimity(\mathcal{I}))\}$.*

We use $num_{(a,b)}(\mathcal{I})$ to denote the cardinality of $\{\pi \in \mathcal{I} : a <_\pi b\}$.

Definition 3. *The* majority graph *of $\mathcal{I}$, denoted by $M(\mathcal{I})$, is a weighted directed graph constructed as follows: for each $a \in U$, we put a vertex in $M(\mathcal{I})$ labeled as a. For each pair of vertices a and b, we put an edge from a to b if $num_{(a,b)}(\mathcal{I}) > num_{(b,a)}(\mathcal{I})$, and set its weight to $num_{(a,b)}(\mathcal{I}) - num_{(b,a)}(\mathcal{I})$.*

Definition 4. *A* tournament majority graph *of $\mathcal{I}$ is a supergraph TM of $M(\mathcal{I})$ whose set of vertices is U, is a tournament, and the weight of any edge in $E(TM) - E(M(\mathcal{I}))$ is zero.*

Dwork et al. observed that σ is in $OPT(\mathcal{I})$ if and only if $E(M(\mathcal{I})) - \sigma$ is a minimum-weight feedback arc set for $M(\mathcal{I})$ [8].

Definition 5. *A subset F of edges of a graph G is called a* feedback arc set *for G if $(E(G) - F) \cup rev(F)$ is transitive.*

It is easy to see that the same observation is true for any tournament majority graph of $\mathcal{I}$:

Observation 2. *A total order σ is in $OPT(\mathcal{I})$ if and only if $E(TM) - \sigma$ is a minimum-weight feedback arc set for a tournament majority TM of $\mathcal{I}$.*

For weighted tournament graphs, the search tree algorithm of Raman and Saurabh [12] can be used to find a minimum-weight feedback arc set of size at most k edges in $O^*(2.415^k)$ time. We should mention that the original algorithm is designed for weighted tournaments with edge weights greater than or equal to one; however, the algorithm can be used for general weights if the search is confined to feedback arc sets that have no more than k edges. This variant is specially useful for finding a minimum-weight feedback arc set in tournament majority graphs, since although these graphs can have zero-weight edges, we will show that they have a minimum-weight feedback arc set with small number of edges.

We use MinFAS(G, k) to refer to this version of the algorithm, shown in the appendix.

Lemma 1. *[12] Suppose that G is a weighted tournament graph and k is a positive integer. Then,* MinFAS(G, k) *returns a minimum-weight feedback arc set of G with at most k edges, if one exists, in time $O^*((1+\sqrt{2})^k) \approx O^*(2.415^k)$.*

Definition 6. *For any multi-set $\mathcal{I}$ with an optimal aggregation σ, $k_t = \tau(\sigma, \mathcal{I})$, $k_{avg} = avg\{\tau(\pi_1, \pi_2) : \pi_1, \pi_2 \in \mathcal{I}\}$, and $k_{max} = max\{\tau(\pi_1, \pi_2) : \pi_1, \pi_2 \in \mathcal{I}\}$.*

Observation 3. $k_t/(m-1) \leq k_{avg} \leq 4k_t/m$.

Proof. By the definition of k_{avg}, there is a total order in $\mathcal{I}$ within the τ-distance $(m-1)k_{avg}$ from $\mathcal{I}$. Since an optimal aggregation cannot have a larger distance from $\mathcal{I}$, $k_t \leq (m-1)k_{avg}$. The triangle inequality proves the second inequality. □

We use $O^*(f(k, |\mathcal{I}|))$ to denote $O(f(k, \mathcal{I}) \cdot |\mathcal{I}|^c)$ for some constant c. In the rest of the paper, we assume that $\mathcal{I}$ is a multi-set on $\mathcal{T}_U$, $|\mathcal{I}| = m \geq 3$, $|U| = n$, and TM is an arbitrary tournament majority graph of $\mathcal{I}$.

3 Parameterized algorithms

3.1 Parameters k_t and k_t/m

We base our analysis on the following lemma; for simplicity, we use d to denote $|E(TM) - unanimous(\mathcal{I})|$ and use k to denote $|E(TM) - \sigma|$ for an arbitrary $\sigma \in OPT(\mathcal{I})$.

Lemma 2. *For any $\pi \in \mathcal{T}_U$,*

$$(d - |E(TM) - \pi|) + |E(TM) - \pi| \cdot \lceil m/2 \rceil \leq \tau(\pi, \mathcal{I})$$

Proof. Each of the k pairs $(a, b) \in E(TM) - \pi$ indicates that π opposes the ordering of $\{a, b\}$ suggested by the majority. Also, by the definition of dirty pairs, for each of the remaining $d - |E(TM) - \pi|$ dirty pairs, there exists a total order in $\mathcal{I}$ that disagrees with the pair's ordering in π. Therefore, the number of disagreements of π with total orders in $\mathcal{I}$ is at least $(d - |E(TM) - \pi|) + |E(TM) - \pi| \cdot \lceil m/2 \rceil$. □

Corollary 1. *For any $\pi \in \mathcal{T}_U$, $|E(TM) - \pi| \leq \tau(\pi, \mathcal{I})/\lceil m/2 \rceil$.*

Corollary 2. $k \leq k_t/\lceil m/2 \rceil$.

Corollary 2 and Lemma 1 prove that we can use MinFas$(TM, k_t/\lceil m/2 \rceil)$ to compute an optimal aggregation in $O^*(2.415^{k_t/\lceil m/2 \rceil})$ time.

Theorem 1. *An optimal aggregation can be found in time* $O^*(2.415^{k_t/\lceil m/2 \rceil}) \leq O^*(2.415^{((m-1)/\lceil m/2 \rceil)k_{avg}}) \leq O^*(2.415^{((m-1)/\lceil m/2 \rceil)k_{max}})$.

We can also use Lemma 2 to improve the $O^*(1.53^{k_t})$ running time by Betzler et al. [4] to $O^*(1.403^{k_t})$. Since $m \geq 3$, Lemma 2 proves the following relationship between d and k:

Corollary 3. $d + k \leq k_t$.

The trick is to use MinFas$(TM, k_t - d)$ for large values of d and brute force search for small values of d.

Theorem 2. *An optimal aggregation can be found in* $O^*(1.475^{k_t})$ *time.*

Proof. If $d \geq \frac{\log_2(1+\sqrt{2})}{1+\log_2(1+\sqrt{2})} k_t$, then by Lemma 1 we can run MinFAS$(TM, k_t - d)$ to obtain an optimal aggregation in time

$$O^*((1+\sqrt{2})^{k_t-d}) \leq O^*((1+\sqrt{2})^{(1-\frac{\log_2(1+\sqrt{2})}{1+\log_2(1+\sqrt{2})})k_t}) < O^*(1.475^{k_t}).$$

Otherwise, if $d < \frac{\log_2(1+\sqrt{2})}{1+\log_2(1+\sqrt{2})} k_t$ we can enumerate all possible orderings of the d dirty pairs to find the optimal aggregation. The running time in this case will be in

$$O^*(2^d) \leq O^*(2^{\frac{\log_2(1+\sqrt{2})}{1+\log_2(1+\sqrt{2})} k_t}) < O^*(1.475^{k_t}).$$

□

In the remainder of this section, we show how to find an optimal aggregation in time $O^*(3^{(d/2)})$. If we use this improvement in the proof of Theorem 2 instead of 2^d, we will get the following.

Theorem 3. *An optimal aggregation can be found in time*

$$O^*(\min\{(1+\sqrt{2})^{k_t-d}, 3^{d/2}\}) \leq O^*(3^{\frac{\log_2(1+\sqrt{2})}{\log_2(3)+2\log_2(1+\sqrt{2})} k_t}) < O^*(1.403^{k_t}).$$

In the following we give a search tree algorithm, shown in Algorithm 1, to find an optimal aggregation. We use C_t to denote a cycle of length t.

The algorithm gradually decides on the orderings of dirty pairs and uses a set L to keep track of the pairs of vertices ordered so far. Each branch is stopped when either all dirty pairs are ordered in L or the computed L does not correspond to any total order.

Algorithm 1: FindAggregation1

Require: $\mathcal{I}$

1 $TM \leftarrow$ a tournament majority graph of $\mathcal{I}$;
2 $O \leftarrow$ EnumAggregations1$(TM, unanimity(\mathcal{I})\})$;
3 **return** $\sigma \in O$ that minimizes $\tau(\sigma, \mathcal{I})$;

Compared to the search tree algorithm of Betzler et al. [4], we incorporate a tournament majority graph into our search algorithm, and branch on triples of dirty pairs that form a C_3 in TM, instead of all triples of dirty pairs. Using the ideas in the search tree algorithm of Raman and Saurabh [12], designed to find a minimum feedback arc set, we go one step further, and consider C_4's whenever possible. Since we will use this search tree algorithm for small values of d, we modify the algorithm of Raman and Saurabh to optimize the running time for small d's. More precisely, in places that they branch on minimal feedback arc sets of a cycle, we branch on all feedback arc sets of the cycle (lines 5 and 7).

Algorithm 2: EnumAggregations1

Require: G, L

1 **if** G does not have a C_3 **then** `/* no cycles remained */`
2 | **return** $E(G)$;
3 **else if** G has a $C_3 = (V_c, E_c)$ with $E_c \cap L \neq \emptyset$ **then**
4 | **if** $E_c \subseteq L$ **then return** $\emptyset$; `/* L has a cycle */`
5 | **else** $P \leftarrow \{\pi \in \mathcal{T}_{V_c} : \pi$ is consistent with $L\}$;
6 **else if** G has a $C_4 = (V_c, E_c)$ **then**
7 | $P \leftarrow \{\pi \in \mathcal{T}_{V_c} : \pi$ is consistent with $L\}$;
8 **else** `/* C3's in G do not have common edges */`
9 | let (V_c, E_c) be a C_3 in G;
10 | let e be a minimum-weight edge in E_c;
11 | $P \leftarrow \{E_c - \{e\} \cup rev(\{e\})\}$;
12 **return** $\bigcup_{\pi \in P}$ EnumAggregations1$((G - rev(\pi)) + \pi, L \cup \pi)$;

Theorem 4. FindAggregation1$(\mathcal{I})$ *returns an optimal aggregation of* $\mathcal{I}$ *in time* $O^*((\sqrt{3})^d)$.

Due to space limitations, the proof of running time of FindAggregation1$(\mathcal{I})$ is moved to appendix.

Algorithm 3: FindAggregation2

Require: $\mathcal{I}$

1 $TM \leftarrow$ a tournament majority graph of $\mathcal{I}$;
2 $k_{max} \leftarrow \max\{\tau(\pi_1, \pi_2) : \pi_1, \pi_2 \in \mathcal{I}\}$;
3 Initialize Q to $E(TM)$; `/* max-weight subset of edges */`
4 **foreach** $\pi \in \mathcal{I}$ **do**
5 **foreach** $S \subseteq (E(TM) - \pi)$ *with* $|S| \leq k_{max}$ **do**
6 $P_1 \leftarrow (E(TM) - \pi) - S$;
7 $P_2 \leftarrow$ MinFAS$(TM - P_1 + rev(P_1), k_{max} - |S|)$;
8 **if** weight$(P_1 \cup P_2) <$ weight(Q) **then** $Q \leftarrow P_1 \cup P_2$;
9 **end**
10 **end**
11 **return** $(E(TM) - Q) \cup rev(Q)$;

Note that for $m \geq 5$ the bound of Theorem 1 is better than $O^*(1.403^{k_t})$, with respect to k_t. Perhaps, the reason is that the value of k_t generally increases when m is increased, and therefore, k_t/m is a more reasonable parameter than k_t for large m's.

3.2 The parameter k_{max}

In this section, we focus on the parameter k_{max} and show how to improve the running time $O^*(2.415^{((m-1)/\lceil m/2 \rceil)k_{max}}) \approx O^*(5.833^{k_{max}})$ to $O^*((4.829)^{k_{max}})$. The idea is to work with a total order in $\mathcal{I}$ that is close to some $\sigma \in OPT(\mathcal{I})$, and agrees with the majority of $\mathcal{I}$ in most pair orderings. The precise algorithm, called FindAggregation2, is shown in Algorithm 3.

The algorithm goes through every $\pi \in \mathcal{I}$. For every π, the algorithm assumes that π is close to some optimal aggregation, and starts the search by deciding on the ordering of the pairs in $E(TM) - \pi$, using the assumption to confine the search space.

Theorem 5. FindAggregation2*($\mathcal{I}$) returns an optimal aggregation in time* $O^*(4.829^{k_{max}})$.

Proof. Since any computed P_2 in line 7 is a feedback arc set for $TM - P_1 + rev(P_1)$, $P_1 \cup P_2$ is always a feedback arc set for TM. By Observation 2, the algorithm is proved to be sound once we show that $P_1 \cup P_2$ is set to a minimum-weight feedback arc set for TM at some point. We suppose that σ is an optimal aggregation. There exists some π in line 4 such that $\tau(\sigma, \pi) \leq$

k_{max}; since otherwise, $mk_{max} < \tau(\sigma, \mathcal{I})$, proving that $mk_{max} < \tau(\omega, \mathcal{I})$ for every $\omega \in \mathcal{I}$, which violates the definition of k_{max}.

The set $(E(TM) - \pi) - (E(TM) - \sigma)$ is among the enumerated S's in line 5, since it is a subset of $\sigma - \pi$ and therefore its size is at most k_{max}. The value of P_1 for this S will be $(E(TM) - \pi) \cap (E(TM) - \sigma)$.

We claim that weight(P_2) will be equal to weight$((E(TM)-\sigma)-P_1)$. The weight of P_2 is not larger than weight$((E(TM) - \sigma) - P_1)$, since $(E(TM) - \sigma) - P_1$ is a feedback arc set for $TM - P_1 + rev(P_1)$ that has no more than $k_{max} - |S|$ edges: the two sets $S = (E(TM) - \pi) - (E(TM) - \sigma)$ and $(E(TM) - \sigma - P_1) = ((E(TM) - \sigma) - (E(TM) - \pi))$ are disjoint subsets of $E(TM)$, and in both sets, each edge connects a pair of vertices that are ordered differently by σ and π. Therefore, $|E(TM) - \sigma - P_1| + |S|$ is at most $\tau(\sigma, \pi)$, which is no more than k_{max}, due to the choice of π.

Consequently, the weight of $P_1 \cup P_2$ is at most weight(P_1)+weight$(E(TM)-\sigma-P_1)$. Since $P_1 \subseteq (E(TM)-\sigma)$, this weight is equal to weight$(E(TM)-\sigma)$. Therefore, $P_1 \cup P_2$ is a minimum-weight feedback arc set in some iteration. This proves that the algorithm is sound.

The FindAggregation2 algorithm can construct the majority graph in $O(mn^2)$ time. Also, computing the value of k_{max} takes at most $O(m^2n^2)$ time. The next time-consuming steps of the algorithm are lines 5 and 7. The first branching step, line 5, takes $O(\binom{|E(TM)-\pi|}{i})$ time, for any $1 \leq i \leq k_{max}$. By Lemma 1, the second branching step, line 7, takes $O^*((2.415)^{k_{max}-i})$ time. Overall, the algorithm runs in time

$$O^*(m^2n^2 + m \cdot \sum_{1 \leq i \leq k_{max}} \binom{|E(TM)-\pi|}{i} \cdot 2.415^{(k_{max}-i)}) =$$
$$O^*(\sum_{1 \leq i \leq k_{max}} \binom{|E(TM)-\pi|}{i} \cdot 2.415^{(k_{max}-i)})$$

By the definition of k_{max}, $\tau(\pi, \mathcal{I}) \leq (m-1)k_{max}$. Hence, Corollary 1 proves that $|E(TM) - \pi| < 2k_{max}$, and the running time is bounded by

$$O^*(\sum_{1 \leq i \leq k_{max}} \binom{2k_{max}}{i} \cdot 2.415^{(k_{max}-i)}) =$$
$$O^*(2.415^{(-k_{max})} \sum_{1 \leq i \leq k_{max}} \binom{2k_{max}}{i} \cdot 2.415^{(2k_{max}-i)}) =$$
$$O^*(2.415^{(-k_{max})} \sum_{1 \leq i \leq 2k_{max}} \binom{2k_{max}}{i} \cdot 2.415^{(2k_{max}-i)}) =$$
$$O^*(2.415^{(-k_{max})}(1 + 2.415)^{2k_{max}}) = O^*(\tfrac{(1+2.415)^2}{2.415}^{k_{max}}) < O^*(4.829^{k_{max}}) \ .$$

□

Theorem 5 improves the best previous running time of $O^*(16^{k_{max}})$ by Betzler et al. [5].

Algorithm 4: EnumAggregations2

```
Require: I
OPT ← ∅;
σ ← FindAggregation1(I);
foreach tournament majority graph TM of I do
    foreach S ⊆ (E(TM) − σ) do
        if |head(S)| = |tail(S)| = |S| then
            P ← {(x, u) : (u, v) ∈ S, x ∈ (interval_σ((v, u)) − head(S))};
            foreach R ⊆ P do
                R ← R ∪ rev(P − R) ∪ S;
                R ← R ∪ {(x, u) : (u, v) ∈ S, (x, v) ∈ R+};
                R ← R ∪ {(v, y) : (u, v) ∈ S, (u, y) ∈ R+};
                Q ← (σ − rev(R+)) ∪ R+;
                if Q is transitive and τ(Q, I) = τ(σ, I) then
                    OPT ← OPT ∪ {Q};
                end
            end
        end
    end
end
return OPT;
```

3.2.1 Enumerating Optimal Aggregations

In this section, we give an algorithm, shown in Algorithm 4, to enumerate $OPT(\mathcal{I})$. The key point is to focus on candidates that are ordered consecutively in the goal optimal aggregation. To this end, we define $seq(\pi)$ for a total order $\pi \in \mathcal{I}$ as $\{(a, b) : \text{ for no } c \in U,\ a <_\pi c <_\pi b\}$, and define $\text{interval}_\pi((u, v))$ for any $(v, u) \in \pi$ as $\{x : x \in U, v <_\pi x <_\pi u\}$. Our algorithm uses the fact that the *seq* of any optimal aggregation is a subset of $M(\mathcal{I})$.

Due to space limitations, the proofs of the following two lemmas are moved to appendix.

Lemma 3. *If there exists a total order $\pi \in \mathcal{T}_U$ that is consistent with R, $seq(\pi) \subseteq E(TM)$ and $seq(\pi) \cap (E(TM) - \sigma) = S$, then $Q = \pi$.*

Lemma 4. *For any fixed TM, $|P| \leq 2k_t/m - |S|$, in line 6.*

Theorem 6. EnumAggregations2($\mathcal{I}$) *returns $OPT(\mathcal{I})$ in time $O^*(36^{k_t/m})$.*

Proof. EnumAggregations2($\mathcal{I}$) iterates through all possible orderings of pairs $\{\{a,b\} : num_{(a,b)}(\mathcal{I}) = num_{(b,a)}(\mathcal{I})\}$. For any fixed ordering L of these pairs, the algorithm searchs for the subset $OPT_L(\mathcal{I}) = \{\pi \in OPT(\mathcal{I}) : \pi \text{ is consistent with } L\}$. It divides $OPT(\mathcal{I})$ further in line 4, and look for the subset $OPT_{L,S}(\mathcal{I})$ of $OPT_L(\mathcal{I})$ defined as $\{\pi \in OPT_L(\mathcal{I}) : seq(\sigma) \cap (E(TM) - \sigma) = S\}$. All potential S's are produced in line 4. Line 5 removes those S's that contain two edges with the same head or the same tail, since the *seq* of a total order cannot contain such edges. Finally, a set P of pairs is computed such that any decision on the orderings of the pairs in P narrows $OPT_{L,S}(\mathcal{I})$ down dramatically. Indeed, Lemma 3 proves that for any chosen R there is either one or zero $\pi \in OPT_{L,S}(\mathcal{I})$ that is consistent with R^+. Furthermore, in case there exists one such π, it is consistent with the transitive relation Q (in line 11). Consequently, we can produce all total orders in $OPT(\mathcal{I})$ by going through all possible R's and see if Q becomes transitive and it is indeed an optimal aggregation.

For any chosen TM, the number of iterations with $|S| = i$, $0 \leq i \leq |E(TM) - \sigma|$, will be $\binom{|E(TM)-\sigma|}{i} \times 2^{|P|}$. We will prove in Lemma 4 that $|P| \leq 2k_t/m - |S|$. Therefore, the number of iterations for each TM is at most $\sum_{0 \leq i \leq |E(TM)-\sigma|} \binom{|E(TM)-\sigma|}{i} \times 2^{2k_t/m-i}$. By Corollary 1, $|E(TM)-\sigma| \leq 2k_t/m$. Therefore, this value is bounded by

$$\sum_{0 \leq i \leq 2k_t/m} \binom{2k_t/m}{i} \times 2^{2k_t/m-i} = 2^{2k_t/m} \times (1 + 1/2)^{2k_t/m} = 9^{k_t/m}$$

Any edge $(a,b) \in L$ indicates that σ opposes the preference of exactly $m/2$ total orders in $\mathcal{I}$. Therefore, $|L| \leq 2k_t/m$. Since there are $2^{|L|}$ possible TM's, the total number of iterations is bounded by $36^{k_t/m}$. □

References

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5):1–27, 2008.

[2] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[3] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Computing Kemeny rankings, parameterized by the average K-T distance. COMSOC '08: 2nd International Workshop on Computational Social Choice, 2008.

[4] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny scores. In *AAIM '08: Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 60–71, 2008.

[5] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. How similarity helps to efficiently compute Kemeny rankings. In *AAMAS '09: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[6] I. Charon and O. Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR*, 5(1):5–60, 2007.

[7] A. Davenport and J. Kalagnanam. A computational study of the Kemeny rule for preference aggregation. In *AAAI '04: Proceedings of the 19th National Conference on Artificial Intelligence*, pages 697–702, 2004.

[8] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW '01: Proceedings of the 10th International Conference on World Wide Web*, pages 613–622, 2001.

[9] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:575–591, 1959.

[10] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.

[11] B. Monjardet. Tournois et ordres médians pour une opinion. *Mathématiques et Sciences humaines*, pages 55–73, 1973.

[12] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[13] D. P. Williamson and A. van Zuylen. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *WAOA '07: Proceedings of the 5th Workshop on Approximation and Online Algorithms*, pages 260–273, 2008.

Appendix

Proof of Theorem 4. We prove that the algorithm runs in $O^*((\sqrt{3})^d)$ time. We use $u(L)$ to denote the number of undecided pairs, i.e. $|\{\{a,b\} : a,b \in U, (a,b) \notin L\}|$. Initially $u(L)$ is d. If at any point $u(L)$ becomes zero, the algorithm returns: either G does not have a cycle at that point, causing a return in line 2, or has some C_3's, in which case all edges of its C_3's are also in L, causing a return in line 4. For non-zero $u(L)$'s, if E_c has two edges in L, the algorithm branches on one case and reduces $u(L)$ by one in line 5. If E_c has one edge in L, the algorithm branches on three cases, reducing $u(L)$ by two in all cases, in line 5. If E_c does not have an edge in L and forms a C_4, then E_c has either zero or one edges in L. Therefore, either the algorithm branches on 24 cases, reducing $u(L)$ by six in all cases, or branches on 12 cases, reducing $u(L)$ by five in all cases, in line 7. Otherwise, E_c has no edges in L and the algorithm branches on one case, reducing $u(L)$ by three, in line 11.

Therefore, the size of the search tree is bounded by $\alpha^{u(L)}$ for any α that satisfies the following inequalities:

$$\begin{aligned}
\alpha^0 &\geq 1 \\
\alpha^{u(L)} &\geq 1 \times \alpha^{u(L)-1} \\
\alpha^{u(L)} &\geq 3 \times \alpha^{u(L)-2} \\
\alpha^{u(L)} &\geq 12 \times \alpha^{u(L)-5} \\
\alpha^{u(L)} &\geq 24 \times \alpha^{u(L)-6} \\
\alpha^{u(L)} &\geq 1 \times \alpha^{u(L)-3}
\end{aligned}$$

Consequently, the running time of the algorithm is in $O^*((\sqrt{3})^d)$. □

To prove Lemma 3, we need two auxiliary lemmas:

Lemma 5. *For any edge $(u,v) \in S$ and any vertex $x \in interval_\sigma((u,v))$, either $(x,u) \in R^+$ or $(x,u) \in rev(R^+)$.*

Proof. We prove the lemma by strong induction: assuming that the statement is true for every x' with $x <_\sigma x'$ that is in an interval of some edge in S, we prove the claim for x.

By definition of P in line 6, the lemma holds for $x \notin head(S)$.

If $x \in head(S)$, there exists an edge $(w,x) \in S$. Due to the condition of line 5, the edges in S do not have common heads, and hence, $w = u$ cannot happen. If $w <_\sigma u$, then $w \in interval_\sigma((u,v))$. Therefore, by induction hypothesis, either $(w,u) \in R^+$ or $(w,u) \in rev(R^+)$. In the former case.

(x, u) is inserted into R in line 10 when S is set to (w, x). In the latter case, since $(u, w) \in R^+$ and $(w, x) \in S \subseteq R$, the transitive closure of R contains (u, x), and hence, $(x, u) \in rev(R^+)$.

If $u <_\sigma w$, then $u \in interval_\sigma((w, x))$. By induction hypothesis, either $(u, w) \in R^+$ or $(u, w) \in rev(R^+)$. In the former case, $(u, w) \in R^+$ and $(w, x) \in S \subseteq R$ result in $(u, x) \in R^+$. In the latter case, (x, u) is inserted into R in line 10 when S is set to (w, x). □

Lemma 6. *Suppose that there exists a total order $\pi \in \mathcal{T}_U$ that is consistent with R. If $x <_\sigma y$ and $y <_\pi x$, then $(y, x) \in R^+$.*

Proof. We prove the lemma by strong induction: assuming that the statement is true for every x' with $x <_\sigma x'$ and every y' that satisfies $x' <_\sigma y'$ and $y' <_\pi x'$, we prove the claim for x.

Suppose that π imposes the order $...w_1 w_2 \dots w_\ell...$, for $w_1 = y$, $w_\ell = x$, and $\ell \geq 2$. Since $x <_\sigma y$, π cannot order x after y unless there exist some $1 \leq i \leq \ell$ such that $(w_i, w_{i+1}) \in S$ with $x = w_{i+1}$ or $x \in interval_\sigma((w_i, w_{i+1}))$.

In both cases $(w_i, x) \in R^+$: in the former case, $w_{i+1} = x$. Therefore, $(w_i, x) \in S \subseteq R^+$. In the latter case, $x \in interval_\sigma((w_i, w_{i+1}))$. By Lemma 5, (x, w_i) is in R^+ or $rev(R^+)$. Since π is consistent with R^+ and orders w_i before x, then (w_i, x) must be in R^+.

If $w_i = y$, then $(y, x) \in R^+$ is trivially true. For other cases, we show that $(y, w_i) \in R^+$, which proves that $(y, x) \in R^+$: if $w_i <_\sigma y$, we can use the induction hypothesis for w_i and y to prove that $(y, w_i) \in R^+$. Otherwise, if $y <_\sigma w_i$ then $y \in interval_\sigma((w_i, w_{i+1}))$. Thus, by Lemma 5 (y, w_i) is in R^+ or $rev(R^+)$. Since π is consistent with R^+ and orders y before w_i, (y, w_i) must be in R^+. □

Proof of Lemma 3. As is true for any total order, $(seq(\pi))^+ = \pi$. Therefore, if $seq(\pi) \subseteq Q$ and Q is transitive, then $Q = \pi$.

We first prove that $seq(\pi) \subseteq Q$. Due to the assumptions for $seq(\pi)$, the edges in $seq(\pi)$ are either in S or in $E(TM) - (E(TM) - \sigma)$, which is a subset of σ. Therefore, $seq(\pi) = S \cup K$ for some $K \subseteq \sigma$. By definitions of R and Q, in lines 8 and 11, $S \subseteq Q$. It only remains to prove that $K \subseteq Q$. Since π is consistent with R, it is also consistent with R^+, which proves that any subset of π, in particular K, has no edges in $rev(R^+)$. As Q contains all edges in σ except for the edges in $rev(R^+)$, it must include all edges in K.

Next, we prove that $Q = (\sigma - rev(R^+)) \cup R^+$ is transitive. Since Q is a complete binary relation, it is enough to show that it does not contain a C_3. Assume instead that there is a C_3 on a set of edges $E_c =$

$\{(a,b),(b,c),(c,a)\} \subseteq Q$. Not all the edges in E_c are in $\sigma - rev(R^+)$, since $\sigma - rev(R^+)$ is part of the transitive binary relation σ. On the other hand, there exists at most one edge in E_c that is in R^+, due to the transitivity of R^+. Consequently, the only potential case is that two of the edges in E_c, say (a,b) and (b,c), are in $\sigma - rev(R^+)$ and the other edge, i.e. (c,a), is in R^+ but not in $\sigma - rev(R^+)$.

Therefore, we know that $a <_\sigma b <_\sigma c$. If $b <_\pi a$, then by Lemma 6 $(b,a) \in R^+$, which contradicts $(a,b) \in \sigma - rev(R^+)$. Otherwise, if $c <_\pi b$, then by Lemma 6 $(c,b) \in R^+$, which contradicts $(b,c) \in \sigma - rev(R^+)$. □

Proof of Lemma 4. The idea is to prove that there are at least $(|P| + |S|)m/2$ tuples of the form $(\{x,y\},\pi)$, $x,y \in U, \pi \in \mathcal{I}$, such that the ordering of $\{x,y\}$ in σ differs from its ordering in π. That gives a lower-bound for k_t.

Every edge $(u,v) \in S$ indicates that σ opposes the preference of u to v in a multi-set $O \subseteq \mathcal{I}$ of at least $m/2$ votes. Moreover, for any w in $\text{interval}_\sigma((u,v)) - head(S)$, each vote in O orders either $\{u,w\}$ or $\{v,w\}$ differently from σ. Hence, for at least $(1 + |\text{interval}_\sigma((u,v)) - head(S)|)m/2$ tuples of the multi-set

$$\{(\{x,y\},\pi) : \pi \in O, ((x = u, y = v) \text{ or } (x \in \{u,v\}, y \in \text{interval}_\sigma((u,v)) - head(S)))\}$$

the ordering of $\{x,y\}$ in σ differs from its ordering in π.

Due to the condition in line 5, different edges $e_1, e_2 \in S$ correspond to disjoint groups of tuples. Indeed, tuples of different edges in S have different first elements, since otherwise, either e_1 and e_2 have a common endpoint and there is a vertex in $\text{interval}_\sigma(e_1) \cap \text{interval}_\sigma(e_2)$, or e_1 has an endpoint in $\text{interval}_\sigma(e_2) - head(S)$ and e_2 has an endpoint in $\text{interval}_\sigma(e_1) - head(S)$. The first case cannot happen, since the edges in S do have common heads or common tails, and therefore, any two edges of S that share an endpoint must have different sets of vertices in their intervals. The second case cannot happen, since otherwise, $tail(e_1) \in \text{interval}_\sigma(e_2)$ and $tail(e_2) \in \text{interval}_\sigma(e_1)$ must hold, which is not possible.

Consequently, the total number of disagreements of σ with the total orders in $\mathcal{I}$, which is $\sum_{\pi \in \mathcal{I}} \tau(\pi,\sigma)$, is at least $\sum_{e \in S}(1 + |\text{interval}_\sigma(e) - head(S)|)m/2$:

$$m/2 \cdot (\sum_{e \in S}(|\text{interval}_\sigma(e) - head(S)|) + |S|) \leq \tau(\sigma,\mathcal{I}).$$

□

Algorithm 5: MinFas

Require: G, Y, k, F_{opt}
; /* initially called with $G, \emptyset, k, E(G)$ */

if G does not have a C_3 **then** /* no cycles remained */
  return $F \in \{F_{opt}, Y\}$ with minimum weight;
else if $|Y| = k$ **then** /* cannot afford more edges */
  return $\emptyset$;
else if G has a $C_3 = (V_c, E_c)$ with $E_c \cap Y \neq \emptyset$ **then**
  if $E_c \subseteq Y$ **then return** $\emptyset$; /* Y has a cycle */
  else $P \leftarrow \{F : F \cap Y = \emptyset, F$ is a minimal subset of E_c such that $E_c - F$ is transitive$\}$;
else if G has a $C_4 = (V_c, E_c)$ **then**
  $P \leftarrow \{F : F \cap Y = \emptyset, F$ is a minimal subset of E_c such that $E_c - F$ is transitive$\}$;
else /* C_3's in G do not have common edges */
  let (V_c, E_c) be a C_3 in G;
  let e be a minimum-weight edge in E_c;
  $P \leftarrow \{e\}$;
return $F \in \{F_{opt}\} \cup \{\text{MinFas}(G - F + rev(F), Y \cup F) : F \in P\}$ with minimum weight;