Proc. 4th AAIM, 2008

Fixed-Parameter Algorithms for Kemeny Scores

Nadja Betzler[1,⋆], Michael R. Fellows[2,⋆⋆], Jiong Guo[1,⋆⋆⋆], Rolf Niedermeier[1], and Frances A. Rosamond[2,⋆⋆]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Germany.
{betzler,guo,niedermr}@minet.uni-jena.de
[2] PC Research Unit, Office of DVC (Research), University of Newcastle,
Callaghan, NSW 2308, Australia.
{michael.fellows,frances.rosamond}@newcastle.edu.au

Abstract. The Kemeny Score problem is central to many applications in the context of rank aggregation. Given a set of permutations (votes) over a set of candidates, one searches for a "consensus permutation" that is "closest" to the given set of permutations. Computing an optimal consensus permutation is NP-hard. We provide first, encouraging fixed-parameter tractability results for computing optimal scores (that is, the overall distance of an optimal consensus permutation). Our fixed-parameter algorithms employ the parameters "score of the consensus", "maximum distance between two input permutations", and "number of candidates". We extend our results to votes with ties and incomplete votes, thus, in both cases having no longer permutations as input.

1 Introduction

To aggregate inconsistent information does not only appear in classical voting scenarios but also in the context of meta search engines and many other applications [8, 6, 1, 5]. In some sense, herein one always deals with *consensus problems* where one wants to find a solution to various "input demands" such that these demands are met as well as possible. Naturally, contradicting demands cannot be fulfilled at the same time. Hence, the consensus solution has to provide a balance between opposing requirements. The concept of *Kemeny consensus* is among the most classical and important research topics in this context. In this paper, we study new algorithmic approaches based on parameterized complexity analysis [7, 10, 13] for computing Kemeny scores and, thus, Kemeny consensus solutions. To describe our results, we start with introducing Kemeny elections.

Kemeny's voting scheme goes back to the year 1959. It can be described as follows. An *election* (V, C) consists of a set V of n votes and a set C of

⋆ Supported by the DFG, research project DARE, GU 1023/1.
⋆⋆ Supported by the Australian Research Council. Work done while staying in Jena as a recipient (MF) of the Humboldt Research Award of the Alexander von Humboldt foundation, Bonn, Germany.
⋆⋆⋆ Supported by the DFG, Emmy Noether research group PIAF, NI 369/4.

m candidates. A vote is a *preference list* of the candidates, that is, for each voter the candidates are ordered according to preference. For instance, in case of three candidates a, b, c, the order $c > b > a$ would mean that candidate c is the best-liked one and candidate a is the least-liked one for this voter.[3] A "Kemeny consensus" is a preference list that is "closest" to the preference lists of the voters: For each pair of votes p, q, the so-called *Kendall-Tau distance* (*KT-distance* for short) between p and q, also known as the number of inversions between two permutations, is defined as $\mathrm{dist}(p,q) = \sum_{\{c,d\}\subseteq C} d_{p,q}(c,d)$, where the sum is taken over all unordered pairs $\{c,d\}$ of candidates, and $d_{p,q}(c,d)$ is 0 if p and q rank c and d in the same order, and 1 otherwise. Using divide and conquer, the KT-distance can be computed in $O(m \cdot \log m)$ time. The *score* of a preference list l with respect to an election (V, C) is defined as $\sum_{v\in V} \mathrm{dist}(l, v)$. A preference list l with the minimum score is called *Kemeny consensus* of (V, C) and its score $\sum_{v\in V} \mathrm{dist}(l, v)$ is the *Kemeny score* of (V, C). The problem considered in this work is as follows:

Kemeny Score
Input: An election (V, C) and a positive integer k.
Question: Is the Kemeny score of (V, C) at most k?

Clearly, in applications we are mostly interested in computing a Kemeny consensus of a given election. All our algorithms that decide the Kemeny Score problem actually provide a corresponding Kemeny consensus.

Known results. We summarize the state of the art concerning the computational complexity of Kemeny Score. Bartholdi et al. [2] showed that Kemeny Score is NP-complete, and it remains so even when restricted to instances with only four votes [8, 9]. Given the computational hardness of Kemeny Score on the one side and its practical relevance on the other side, polynomial-time approximation algorithms have been studied. Thus, the Kemeny score can be approximated to a factor of 8/5 by a deterministic algorithm [16] and to a factor of 11/7 by a randomized algorithm [1]. Recently, a (theoretical) PTAS result has been obtained [12]. Conitzer, Davenport, and Kalagnanam [6, 5] performed computational studies for the efficient exact computation of a Kemeny consensus, using heuristic approaches such as greedy and branch and bound. Finally, note that Hemaspaandra et al. [11] provided further, exact classifications of the computational complexity of Kemeny elections.

Our results. As pointed out by Conitzer et al. [5], for obvious reasons approximate solutions for election problems such as Kemeny Score may be of limited interest. Hence, *exact* solutions are of particular relevance in this context. Given the NP-completeness of the problem, however, it seems inevitable to live with exponential-time algorithms for solving Kemeny Score. Fortunately, parameterized complexity analysis as pioneered by Downey and Fellows [7, 10, 13] seems a fruitful approach here. This will be shown by positive results based on three natural parameterizations. Before that, note that studying the parameter "number of votes" is pointless because, as mentioned before,

[3] Some definitions also allow ties between candidates—we deal with this later.

the problem is already NP-complete for only four votes. First of all, using the Kemeny score k itself as the parameter, we derive an algorithm solving Kemeny Score in $O(1.53^k + m^2 n)$ time, where $n := |V|$ and $m := |C|$. This algorithm is based on a problem kernelization and a depth-bounded search tree. Further, we introduce a structural parameterization by studying the parameter "maximum KT-distance d between any two input votes". Note that in application scenarios such as meta search engines small d-values may be plausible. We show that Kemeny Score can be solved in $O((3d+1)! \cdot d \cdot \log d \cdot m \cdot n)$ time by a dynamic programming approach. Eventually, note that by trying all possible permutations of the m candidates, we can trivially attain an efficient algorithm if m is very small. The corresponding combinatorial explosion $m!$ in the parameter m is fairly large, though. Using dynamic programming, we can improve this to an algorithm running in $O(2^m \cdot m^2 \cdot n)$ time. Finally, we extend our findings to the cases where ties within votes are allowed and to incomplete votes where not all candidates are ranked. Due to the lack of space, several proofs are deferred to the full version.

2 Preliminaries

We refer to the introductory section for some basic definitions concerning (Kemeny) elections. Almost all further concepts are introduced where needed. Hence, here we restrict ourselves to concepts of "general" importance. Let the *position* of a candidate a in a vote v be the number of candidates that are better than a in v. That is, the leftmost (and best) candidate in v has position 0 and the rightmost has position $m-1$. Then, $\mathrm{pos}_v(a)$ denotes the position of candidate a in v. Moreover, we say that two candidates a and b form a *dirty pair* if in V there is one vote with $a > b$ and another vote with $b > a$.

We briefly introduce the relevant notions of parameterized complexity theory [7, 10, 13]. Parameterized algorithmics aims at a multivariate complexity analysis of problems. This is done by studying relevant problem parameters and their influence on the computational complexity of problems. The hope lies in accepting the seemingly inevitable combinatorial explosion for NP-hard problems, but confining it to the parameter. Hence, the decisive question is whether a given parameterized problem is *fixed-parameter tractable (FPT)* with respect to the parameter, say k. In other words, here we ask for the existence of a solving algorithm with running time $f(k) \cdot \mathrm{poly}(n, m)$ for some computational function f. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction rules*, often yielding a *kernelization.* Herein, the goal is, given any problem instance x with parameter k, to transform it in polynomial time into a new instance x' with parameter k' such that the size of x' is bounded from above by some function only depending on k, $k' \leq k$, and (x, k) is a yes-instance if and only if (x', k') is a yes-instance. We call a data reduction rule *sound* if the new instance after an application of this rule is a yes-instance iff the original instance is a yes-instance. We also employ search trees for our fixed-parameter algorithms. Search tree algorithms work in a recursive manner.

The number of recursion calls is the number of nodes in the according tree. This number is governed by linear recurrences with constant coefficients. It is well known how to solve these [13]. If the algorithm solves a problem instance of size s and calls itself recursively for problem instances of sizes $s - d_1, \dots, s - d_i$, then $(d_1, \dots, d_i)$ is called the *branching vector* of this recursion. It corresponds to the recurrence $T_s = T_{s-d_1} + \cdots + T_{s-d_i}$ for the asymptotic size T_s of the overall search tree.

3 Parameterization by the Kemeny Score

We present a kernelization and a search tree algorithm for Kemeny Score. The following lemma, whose correctness follows directly from the Extended Condorcet criterion [15], is used for deriving the problem kernel and the search tree.

Lemma 1. *Let a and b be two candidates in C. If $a > b$ in all votes $v \in V$, then every Kemeny consensus has $a > b$.*

3.1 Problem Kernel

When applied to an input instance of Kemeny Score, the following polynomial-time executable data reduction rules yield an "equivalent" election with at most $2k$ candidates and at most $2k$ votes with k being the Kemeny score. Note that, if we use a preference list over a *subset* of the candidates to describe a vote, then we mean that the remaining candidates are positioned arbitrarily in this vote. We apply the following data reduction rule to shrink the number of candidates in a given election (V, C).

Rule 1. Delete all candidates that are in no dirty pair.

Lemma 2. *Rule 1 is sound and can be carried out in $O(m^2 n)$ time.*

Lemma 3. *After having exhaustively applied Rule 1, in a yes-instance there are at most $2k$ candidates.*

Next, we apply a data reduction rule to get rid of too many identical votes.

Rule 2. If there are more than k votes in V identical to a preference list l, then return "yes" if the score of l is at most k; otherwise, return "no".

Lemma 4. *Rule 2 is sound and works in $O(mn)$ time.*

Lemma 5. *After having exhaustively applied Rule 1 and Rule 2, in a yes-instance $((V, C), k)$ of* Kemeny Score *there are at most $2k$ votes.*

In summary, we can state the following:

Theorem 1. Kemeny Score *admits a problem kernel with at most $2k$ votes over at most $2k$ candidates. It can be computed in $O(m^2 \cdot n)$ time.*

3.2 Search Tree Algorithm

It is trivial to achieve an algorithm with search tree size $O(2^k)$ by simply branching on dirty pairs. For the description of an improved search tree algorithm, we need the following definition: Three candidates a, b, c form a *dirty triple* if they occur in at least two dirty pairs. The search tree algorithm first enumerates all dirty pairs of the given election (V, C) and then branches according to the dirty triples. At a search tree node, in each case of the branching, an order of the candidates involved in the dirty triples processed at this node is fixed and maintained in a set. This order represents the relative positioning of these candidates in the Kemeny consensus sought for. Then, the parameter is decreased according to this order. Since every order of two candidates in a dirty pair decreases the parameter at least by one, the height of the search tree is upper-bounded by the parameter.

Next, we describe the details of the branching. At each node of the search tree, we store two types of information:

– The dirty pairs that have not been processed by ancestor nodes are stored in a set D.
– The information about the orders of candidates that have already been determined when reaching the node is stored in a set L. That is, for every pair of candidates whose order is already fixed we store this order in L.

For any pair of candidates a and b, the order $a > b$ is *implied* by L if there is a subset of ordered pairs $\{(c_1, c_2), (c_2, c_3), \ldots, (c_{i-1}, c_i)\}$ in L which can be concatenated such that we have $a > c_1 > \cdots > c_i > b$. To add the order of a "new" pair of candidates, for example $a > b$, to L, we must check if this is *consistent* with L, that is, L does not already imply $b > a$.

At the root of the search tree, D contains all dirty pairs occurring in (V, C). For each non-dirty pair, its relative order in an optimal Kemeny ranking can be determined using Lemma 1. These orders are stored in L. At a search tree node, we distinguish three cases:

Case 1. If there is a dirty triple $\{a, b, c\}$ forming three dirty pairs contained in D, namely, $\{a, b\}, \{b, c\}, \{a, c\} \in D$, then remove these three pairs from D. Branch into all six possible orders of a, b, and c. In each subcase, if the corresponding order is not consistent with L, discard this subcase, otherwise, add the corresponding order to L and decrease the parameter according to this subcase. The worst-case branching vector of this case is $(3, 4, 4, 5, 5, 6)$, giving a branching number 1.52. To see this, note that we only consider instances with at least three votes, since Kemeny Score is polynomial-time solvable for only two votes. Thus, for every dirty pair $\{c, c'\}$, if there is only one vote with $c > c'$ (or $c' > c$), then there are at least two votes with $c' > c$ (or $c > c'$). A simple calculation then gives the branching vector.

Case 2. If Case 1 does not apply and there is a dirty triple $\{a, b, c\}$, then a, b, c form exactly two dirty pairs contained in D, say $\{a, b\} \in D$ and $\{b, c\} \in D$. Remove $\{a, b\}$ and $\{b, c\}$ from D. As $\{a, c\}$ is not a dirty pair, its order is determined by L. Hence, we have to distinguish the following two subcases.

If $a > c$ is in L, then branch into three further subcases, namely,

- $b > a > c$,
- $a > b > c$, and
- $a > c > b$.

For each of these subcases, we add the pairwise orders induced by them into L if they are consistent for all three pairs and discard the subcase, otherwise. The worst-case branching vector here is $(3, 3, 2)$, giving a branching number 1.53.

If $c > a$ is in L, then we also get the branching vector $(3, 3, 2)$ by branching into the following three further subcases:

- $b > c > a$,
- $c > b > a$, and
- $c > a > b$.

Case 3. If there is no dirty triple but at least one dirty pair (a, b) in D, then check whether there exists some relative order between a and b implied by L: If L implies no order between a and b, then add an order between a and b to L that occurs in at least half of the given votes; otherwise, we add the implied order to L. Finally, decrease the parameter k according to the number of votes having a and b oppositely ordered compared to the one added to L.

The search tree algorithm outputs "yes" if it arrives at a node with $D = \emptyset$ and a parameter value $k \geq 0$; otherwise, it outputs "no". Observe that a Kemeny consensus is then the *full* order implied by L at a node with $D = \emptyset$.

Combining this search tree algorithm with the kernelization given in Section 3.1, we arrive at the main theorem of this section.

Theorem 2. Kemeny Score *can be solved in* $O(1.53^k + m^2 n)$ *time, where* k *denotes the Kemeny score of the given election.*

Observe that the search tree algorithm also works for instances in which the votes are weighted by positive integers. If the votes have arbitrary positive weights, we can use the dynamic programming algorithm that is described in the following section.

4 Parameterization by the Maximum KT-distance

The Kemeny score of some instances might be quite large. So, we consider a further parameterization, now with the maximum KT-distance d between any two given votes as the parameter. Formally, $d := \max_{p,q \in V}\{\text{dist}(p, q)\}$. We describe a dynamic programming fixed-parameter algorithm for this parameterization. To this end, we need the following definitions:

A *block* of size s with start position p denotes the set of candidates a satisfying $p \leq \text{pos}_v(a) \leq p + s - 1$ for at least one $v \in V$. By block(p) we denote a block of size $d+1$ with start position p. With a *subvote* $v_{C'}$ of a vote v restricted to a subset $C' \subseteq C$ we mean the order of the candidates in C' such that the preferences of v with respect to the candidates in C' are preserved. A *subinstance* of (V, C) restricted to $C' \subseteq C$ consists of all $v_{C'}$ with $v \in V$.

Now, we state two lemmas that are crucial for the algorithm.

Lemma 6. *In an input instance with maximum KT-instance d, the positions of a candidate in two votes differ by at most d.*

Proof. Consider a candidate a and two votes v and w. Assume that $\text{pos}_v(a) = p$ and $\text{pos}_w(a) \geq p + d + 1$. Then, in w there are at least $p + d + 1$ candidates that are better than a, and in v there are p candidates that are better than a. Therefore, there are at least $d+1$ candidates that are ranked higher than a in w and lower than a in v. This means that the KT-distance between v and w is at least $d + 1$, a contradiction. □

Lemma 7. *In an input instance with maximum KT-instance d, every block of size $d + 1$ contains at most $3d + 1$ candidates.*

Proof. Assume that there are $3d + 2$ candidates $c_1, c_2, \ldots, c_{3d+2}$ in a block of size $d + 1$ with start position p. Without loss of generality, assume that $v_1 := c_1 > \cdots > c_d > c_{d+1} > \cdots > c_{3d+2}$. Let c_i be the candidate in position p in v_1. If $i \leq d + 1$, then candidate c_{3d+2} is in v_1 in position $p + 2d + 1$ or in a higher position. However, since c_{3d+2} is in the block with start position p, it must occur in position $p+d$ or lower in some other vote, contradicting Lemma 6. If $i > d+1$, then a similar argument applies to c_1, again contradicting Lemma 6. □

Basic idea. For designing the algorithm we exploit that blocks of size $d + 1$ can be used to decompose an election when computing the Kemeny score. More precisely, consider a block(i) of size $d+1$, then any candidate a with $a \notin \text{block}(i)$ must fulfill either $\text{pos}_v(a) < i$ for all $v \in V$ or $\text{pos}_v(a) > i + d + 1$ for all $v \in V$. Further, we can show that there is a Kemeny consensus in which a must have a position in a range that is a function of d. This means that, if we iterate from left to right over the votes and store the Kemeny score for all "partial orders" of the candidates of a block, then we can forget all candidates that appear only left of this block.

Roughly speaking, our algorithm works as follows: It iterates from left to right over the votes and, in each iterative step, considers a block of size $d + 1$. That is, in the first iterative step, the initialization, it considers all possible orders of the candidates of block(0) and uses a table to store the scores of all such orders with respect to the subinstance of (V, C) restricted to block(0). Then, in the following iterations, it considers block(i) with $i \geq 1$. The computation of the score of the orders of block(i) is based on the table entries for block$(i - 1)$. The Kemeny score of (V, C) is the minimum of the entries for block$(m - d - 1)$. For the formal description of the algorithm we need some further definitions.

Definitions. For a subset of candidates C', let $\pi(C')$ denote an order of C'. For two candidate subsets C' and C'', the orders $\pi'(C')$ and $\pi''(C'')$ are *compatible* if the candidates of $C' \cap C''$ have the same order according to π' and π''. Further,

Initialization:
For all permutations $\pi(\text{block}(0))$

$$T(\pi(\text{block}(0)), 0) := S(\pi(\text{block}(0)))$$

Update:
For $i = 1, \ldots, m-d-1$

$$T(\pi(\text{block}(i)), i) := \min_{\pi'(\text{block}(i-1)) \in \text{Comp}(i,\pi)} \{T(\pi'(\text{block}(i-1)), i-1)\} + S(\pi(\text{block}(i))) - S(\pi_{\text{block}(i-1)}(\text{block}(i)))$$

Output:

$$\min_{\pi(\text{block}(m-d-1))} \{T(\pi(\text{block}(m-d-1)), m-d-1)\}.$$

Fig. 1. Dynamic programming algorithm for Kemeny Score.

if $C' \subseteq C''$, let $\pi_{C'}(C'')$ denote the suborder of $\pi(C'')$ restricted to C'. For an order $\pi(\text{block}(i))$ the set $\text{Comp}(i,\pi)$ contains all orders of candidates of $\text{block}(i-1)$ that are compatible with $\pi(\text{block}(i))$. For any fixed order $\pi(C')$ of $C' \subseteq C$, its *score* $S(\pi(C'))$ is the sum of its KT-distances between $\pi(C')$ and all subvotes of votes in V restricted to C'.

Algorithmic details. Next, we define the table T used in the dynamic programming algorithm. The columns of T one-to-one correspond to the candidate subsets $\text{block}(i) \subseteq C$ with $0 \leq i \leq m-d-1$. Each row corresponds to a possible order of the candidates in $\text{block}(i)$. By Lemma 7, the number of candidates of each block is upper-bounded by $3d+1$. Thus, T has at most $(3d+1)!$ rows. An entry in column $\text{block}(i)$ and row π, where π is an order of the candidates in $\text{block}(i)$, stores the minimum score of the orders π' of $C' := \bigcup_{j \leq i} \text{block}(j)$ under the condition that π' is compatible with π.

The algorithm is given in Figure 1. In the following, we first state the correctness of the dynamic programming and then analyze its running time.

Lemma 8. *The dynamic programming algorithm (see Figure 1) correctly decides* Kemeny Score.

Theorem 3. Kemeny Score *can be solved in* $O((3d+1)! \cdot d \cdot \log d \cdot m \cdot n)$ *time with* d *being the maximum KT-distance between two input votes.*

Proof. As argued above, the table size is bounded by $O((3d+1)! \cdot m)$. For the initialization, we compute the score of all orders of at most $3d+1$ candidates and, for each order, the score is computable in $O(d \log d \cdot n)$ time as argued in the introductory section. Therefore, the initialization can be done in $O((3d+1)! \cdot d \log d \cdot n)$ time.

At $\text{block}(i)$ with $i \geq 1$, we compute

$$\min_{\pi'(\text{block}(i-1)) \in \text{Comp}(i,\pi)} \{T(\pi'(\text{block}(i-1)), i-1)\}$$

for all possible orders $\pi(\text{block}(i))$. Using a pointer data structure, this can be done in $(3d+1)! \cdot d \log d$ time by going through the $(i-1)$th column once. Thus, the entries of $T(\pi(\text{block}(i)), i)$ can be computed in $O((3d+1)! \cdot d \log d \cdot n)$ time, giving the running time of the algorithm. $\square$

Note that it is easy to modify the dynamic programming to return a Kemeny consensus within the same asymptotic running time. Furthermore, the algorithm can be easily adapted to deal with weighted votes and/or weighted candidates.

5 Parameterization by the Number of Candidates

Theorem 4. Kemeny Score *can be solved in* $O(2^m \cdot m^2 \cdot n)$ *time.*[4]

Proof. (Sketch) The algorithm goes like this: For each subset $C' \subseteq C$ compute the Kemeny score of the given election system restricted to C'. The recurrence for a given subset C' is to consider every subset $C'' \subseteq C'$ where C'' is obtained by deleting a single candidate c from C'. Let π'' be a Kemeny consensus for the election system restricted to C''. Compute the score of the permutation π' of C' obtained from π'' by putting c in first position. Take the minimum score over all π' obtained from subsets of C'. The correctness of this algorithm follows from the following claim.
Claim: A permutation π' of minimum score obtained in this way is a Kemeny consensus of the election system restricted to C'.
Proof: Let σ' be a Kemeny consensus of the election system restricted to C', and suppose that candidate c is on the first position of σ'. Consider $C'' = C' \setminus \{c\}$, and let σ'' be the length-$|C''|$ tail of σ'. The score of σ' is $t + u$, where u is the score of σ'' for the votes that are the restriction of V to C'', and where t is the "cost" of inserting c into the first position: Herein, the cost is the sum over the votes (restricted to C') of the distance of c from first position in each vote. Now suppose that π'' is a Kemeny consensus of the election system restricted to C''. Compared with the score of σ', augmenting π'' to π' by putting c in the first position increases the score of π' by exactly t. The score of π'' is at most u (since it is a Kemeny consensus for the election system restricted to C''), and so the score of π' is at most $t + u$, so it is a Kemeny consensus for the election system restricted to C'. This completes the proof of the claim.

For each of the subsets of C, the algorithm computes a Kemeny score. Herein, a minimum is taken from at most m values and the computation of each of these values needs $O(m \cdot n)$ time. Therefore, the total running time is $O(2^m \cdot m^2 \cdot n)$. $\square$

6 Ties and Incomplete Votes

The algorithm from Section 5 also applies to the following generalizations; we omit the respective technical details.

[4] We give a direct proof here. Basically the same result also follows from a reduction to Feedback Arc Set on Tournaments and a corresponding exact algorithm [14].

6.1 Kemeny Score with Ties

Introducing ties, we use the definition of Hemaspaandra et al. [11]: Each vote consists of a *ranking* of candidates that can contain ties between candidates and the Kemeny consensus can contain ties as well. In the definition of the KT-distance, we then replace $d_{p,q}(c,d)$ by

$$t_{p,q}(c,d) := \begin{cases} 0 \text{ if } p \text{ and } q \text{ agree on } c \text{ and } d, \\ 1 \text{ if } p\ (q) \text{ has a preference among } c \text{ and } d \text{ and } q\ (p) \text{ does not }, \\ 2 \text{ if } p \text{ and } q \text{ strictly disagree on } c \text{ and } d. \end{cases}$$

Additionally, we extend the KT-distance to candidate weights, that is, given a weight function $w : C \to R^+$, the KT-distance $\operatorname{dist}(p,q)$ between two votes p,q is defined as

$$\operatorname{dist}(p,q) = \sum_{\{c,d\} \subseteq C} w(c) \cdot w(d) \cdot t_{p,q}(c,d).$$

By increasing the Kemeny score by two if two votes strictly disagree on two candidates, the overall score becomes larger. It is not surprising that also in case of ties we can get a linear problem kernel and a search tree with respect to the parameter Kemeny score. To this end, we extend the notion of dirty pairs such that a pair of candidates $\{a,b\}$ is also dirty if we have $a = b$ in one vote and $a > b$ or $a < b$ in another vote. Then, we can obtain a linear problem kernel in complete analogy to the case without ties. Concerning the search tree, by branching for a dirty pair $\{a,b\}$ into the three cases $a > b$, $a = b$, and $a < b$, we already achieve the branching vector $(1,2,4)$ and, hence, the branching number 1.76. For example, having three votes in which candidates a and b are ranked equal in two of the votes and we have $a < b$ in the other vote, by branching into the case "$a = b$" we have one inversion that counts one, branching into "$a < b$" we have two inversions that count one, and branching into "$a > b$" we have two inversions counting one and one inversion counting two.

With some more effort, we can also show that Kemeny Score with Ties is fixed-parameter tractable with respect to the parameter maximum KT-distance. To use the dynamic programming of Section 4, we need some bounds on the positions a candidate can take and the size of a block in analogy to Lemma 6 and Lemma 7. In order to achieve this, we need a preprocessing step, which is based on the following lemma:

Lemma 9. *Let d denote the maximum KT-distance of an input instance. If there is a vote in which at least $d+2$ candidates $c_1, \ldots, c_{d+2}$ are tied, then in a yes-instance the candidates $c_1, \ldots, c_{d+2}$ are tied in all votes.*

Now, we can state the following data reduction rule:

Rule 3. If there are $d+i$ candidates for any integer $i \geq 2$ that are tied in all votes, replace them by one candidate whose weight is the sum of the weights of the replaced candidates.

The correctness of Rule 3 follows from Lemma 9. Further, note that the dynamic programming procedure from Section 4 can be adapted to deal with candidate weights. Now, similar to Lemmas 6 and 7 we can prove the following.

Lemma 10. *In a candidate-weighted instance with maximum KT-instance d obtained after exhaustively applying Rule 3, the positions of a candidate in two votes differ by at most $2d + 1$.*

Lemma 11. *In a reduced instance with maximum KT-instance d, every block of size $2d$ contains at most $6d + 2$ candidates.*

Using Lemmas 10 and 11, we can show that with the dynamic programming algorithm given in Section 4, the following running time can be obtained.

Theorem 5. Kemeny Score with Ties *can be solved in $O((6d+2)! \cdot d \cdot \log d \cdot n \cdot m)$ time with d being the maximum KT-distance between two input votes.*

6.2 Incomplete Votes

We now consider Kemeny Score with Incomplete Votes, first introduced by Dwork et al. [8]. Here, the given votes are not required to be permutations of the entire candidate set, but only of candidate subsets, while the Kemeny consensus sought for should be a permutation of all candidates. In the definition of the KT-distance, we then replace $d_{p,q}(c, d)$ by

$$d'_{p,q}(c,d) := \begin{cases} 0 & \text{if } \{c,d\} \not\subseteq C_p \text{ or } \{c,d\} \not\subseteq C_q \text{ or } p \text{ and } q \text{ agree on } c \text{ and } d, \\ 1 & \text{otherwise,} \end{cases}$$

where C_v contains the candidates occurring in vote v.

A simple reduction from Feedback Arc Set shows that, in contrast to Kemeny Score with complete votes, the parameterization by the maximum KT-distance does not lead to fixed-parameter tractability for Kemeny Score with Incomplete Votes:

Theorem 6. Kemeny Score with Incomplete Votes *is NP-hard even if the maximum KT-distance between two input votes is zero.*

By way of contrast, we now show that, parameterized by the Kemeny score k, also Kemeny Score with Incomplete Votes is fixed-parameter tractable.

Theorem 7. Kemeny Score with Incomplete Votes *is solvable in $(1.48k)^k \cdot p(n, m)$ time with k being the Kemeny score and p being a polynomial function of n and m.*

Proof. Let $((V, C), k)$ be the given Kemeny Score instance. The algorithm consists of three steps. First, transform the given election system into one where each vote contains only two candidates. For example, a vote $a > b > c$ is replaced by threes votes: $a > b$, $b > c$, and $a > c$. Second, find all dirty pairs and, for each of them, remove the corresponding votes and branch into two cases, in each case an order between the two candidates of this pair is fixed and the parameter k is decreased accordingly. Third, after having processed all dirty pairs, we construct an edge-weighted directed graph for each of the election systems generated by the second step. The vertices of this graph one-to-one correspond to the candidates. An arc (c, c') from vertex c to vertex c' is added if there is a vote of the form $c > c'$ and a weight equal to the number of the $c > c'$-votes is assigned to this arc. For each pair of candidates c and c' where the second step has already fixed an order $c > c'$, add an arc (c, c') and assign a weight equal to $k + 1$ to this arc. Observe that solving Kemeny Score on the instances generated by the first two steps is now equivalent to solving the Weighted Directed Feedback Arc Set problems on the constructed edge-weighted directed graphs. Directed Feedback Arc Set is the special case of Weighted Feedback Arc Set with unit edge weight. The fixed-parameter algorithm by Chen et al. [4] for Directed Feedback Arc Set can also handle Weighted Directed Feedback Arc Set with integer edge weights [3]. Therefore, applying this algorithm in the third step gives a total running time of $(1.48k)^k \cdot p(n, m)$ for a polynomial function p. $\square$

References

1. N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. In *Proc. 37th STOC*, pages 684–693. ACM, 2005.
2. J. Bartholdi III, 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*, 6:157–165, 1989.
3. J. Chen. Personal communication. December 2007.
4. J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *Proc. 40th STOC*. ACM, 2008.
5. V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proc. 21st AAAI*, pages 620–626, 2006.
6. A. Davenport and J. Kalagnanam. A computational study of the Kemeny rule for preference aggregation. In *Proc. 19th AAAI*, pages 697–702, 2004.
7. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
8. C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proc. WWW*, pages 613–622, 2001.
9. C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited, 2001. Manuscript.
10. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
11. E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349:382–391, 2005.
12. C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *Proc. 39th STOC*, pages 95–103. ACM, 2007.
13. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
14. V. Raman and S. Saurabh. Improved fixed parameter tractable algorithms for two "edge" problems: MAXCUT and MAXDAG. *Information Processing Letters*, 104(2):65–72, 2007.
15. M. Truchon. An extension of the Condorcet criterion and Kemeny orders. Technical report, cahier 98-15 du Centre de Recherche en Économie et Finance Appliquées, 1998.
16. A. van Zuylen and D. P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *Proc. 5th WAOA*, volume 4927 of *LNCS*, pages 260–273. Springer, 2007.