On the Parameterized Complexity of Minimax Approval Voting

Neeldhara Misra
Department of Computer Science and Automation
Indian Institute of Science, Bangalore, India.
mail@neeldhara.com

Arshed Nabeel
Department of Computer Science and Automation
Indian Institute of Science, Bangalore, India.
arshed.nabeel@gmail.com

Harman Singh
BITS Pilani-K.K. Birla Goa Campus.
harman28@gmail.com

ABSTRACT

In this work, we initiate a detailed study of the parameterized complexity of Minimax Approval Voting. We demonstrate that the problem is W[2]-hard when parameterized by the size of the committee to be chosen, but does admit a FPT algorithm when parameterized by the number of strings that is more efficient than the previous ILP-based approaches for the problem. We also consider several combinations of parameters and provide a detailed landscape of the parameterized and kernelization complexity of the problem. We also study the version of the problem where we permit outliers, that is, where the chosen committee is required to satisfy a large number of voters (instead of all of them). In this context, we strengthen an APX-hardness result in the literature, and also show a simple but strong W-hardness result.

1. INTRODUCTION

Aggregating preferences of agents is a fundamental problem in artificial intelligence and social choice [10]. The typical setting is the following: agents (or voters) express their preferences over alternatives (or candidates), and subsequently, a voting rule selects a winner or a set of winners based on these preferences.

A substantial fragment of research in computational social choice has been devoted to single-winner choice problems (sometimes admitting the possibility of ties, resulting in a collection of co-winners). However, there has been an emerging interest in the algorithmic aspects of multi-winner elections, where the goal is to elect a committee of size k, where k is fixed in advance. In other words, the goal is to determine a set of k "winners" based on an appropriate voting rule. Multi-winner problems have several important applications, such as the election of legislatures and committees using proportional representation. They are also heavily used in resource allocation problems, determining the top few movies, books, or products to be fed into recommendation systems, and so on. In a classroom setting (especially online, such as in a MOOC), using peer reviews to determine the best possible TA team of size, say, ten for a future edition of the course is also a scenario for multi-winner elections.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

Approval Voting. This work is set in the framework of *approval voting systems*, where each voter may select and support at most some small number of candidates [6]. In such a system, each voter determines, for every single candidate, if he approves of him or not. A result is then obtained by applying a predefined election rule to the set of collected votes. We refer to such a collection of votes as an *approval ballot*. In contrast, *multi-winner voting rules*, also known as *choose-k rules*, use the standard election setup where every vote is a total order (or a full ranking) over the set of candidates, and the voting rule returns a collection of possible committees that are tied-for-winning [13]. Although there are connections between the two formats, as we will observe in a moment, our focus will be on the former setup.

Given an approval ballot $\mathcal{V} = \{v_1, \ldots, v_n\}$ that seeks to form a committee C of size k, there can be several measures for how well a particular committee performs with respect to the given ballot. Two such fundamental measures are:

- **Approval Voting.** Here, we seek to minimize the sum of the Hamming distances between C and v_i.
- **Minimax Approval Voting.** Here, we seek to minimize the maximum Hamming distance between C and any v_i.

Note that approval voting amounts to choosing the k "most popular" candidates. If every voter approved a subset of size k, then this would amount to a multi-winner extension of the k-approval rule. We note that there have been several other measures considered in this setting, for example Satisfaction Approval, Proportional Approval, Reweighted approval, and so on. We refer the reader to [2] for some very recent work on these aspects of approval voting.

While approval voting has the advantage of being a rule where the winning committee is easy to compute, it can suffer from ignoring the preferences of many voters. For example, consider a ballot where a Y, which is some subset of k candidates, is approved by a subset X of voters. Note that if $|X|$ is even a little over half the total number of voters, then the committee Y will be a winning committee irrespective of the structure of the remaining votes. In such a situation, the minimax approach to approval voting tries to account for the opinion of every voter in its definition. On the other hand, note that the minimax approval voting rule can sometimes try too hard when satisfying every voter

— it is possible that when, say, a large fraction of the ballot is accounted for, there exists a consensus with a small maximum distance threshold, while accounting for everyone pushes up the threshold by orders of magnitude. Therefore, a natural notion to incorporate into the problem to make it more robust in practice is that of *outliers*. We introduce and study this version of Minimax Approval Voting, which we believe has not been examined before, where we seek a committee of size k that has a Hamming distance of at most d from at least s votes. The original problem is the special case when $s = |\mathcal{V}|$.

Closest String. The Minimax Approval Voting problem is quite similar to the Closest String problem, which is an intensely studied problem in the literature of string algorithms and bioinformatics. Most of our work builds on the work of [15] and [3] that explore the Closest String problem from a parameterized perspective. In Closest String, we are given a set of strings $\{s_1, \ldots, s_n\}$ and the goal is to find a string s that has a small Hamming distance from all the given strings. Note that Minimax Approval Voting is the Closest String problem restricted to a binary alphabet, and accompanied with the additional constraint that the output string have exactly k ones. We note that our proposal for Minimax Approval Voting with outliers is inspired from the analogous question in the context of strings, namely Closest to Most Strings, which is also a well-studied variant [5].

Our Framework. Our focus in this work is on the computational complexity of Minimax Approval Voting and several of its variations. We mostly use the paradigm of *parameterized complexity* [12, 19] but we also explore the hardness of approximation in suitable settings.

One of the fundamental types of parameterized algorithms is *kernelization*, where the main goal is *instance compression* - the objective is to output a smaller instance while maintaining equivalence. When outlining nine important future directions of research in computational social choice in a parameterized setting, one of the questions that emerged was the following [7]:

What is the kernelization complexity of fixed-parameter tractable voting problems with respect to the number m of alternatives, the number n of voters, or some parameter less than m or n? Can we derive polynomial (or even linear) problem kernels for some voting problems with the above parameters?

In this work we address several questions in the context of kernelization, hoping to demonstrate some progress on this theme. Another key challenge proposed in [7] is also regarding the use of ILP-based approaches:

Can the [...] ILP-based fixed-parameter tractability results be replaced by direct combinatorial (avoiding ILPs) fixed-parameter algorithms?

While the best known algorithm for Closest String when parameterized by the number of strings was based on an ILP formulation, we give an argument here for its MAV analog that relies on the framework of Color Coding [1], which provides a completely different perspective, and we hope that our style of application will be of general interest.

Our Contributions and Related Work. We consider the Minimax Approval Voting problem and its variation where we allow for outliers, from a parameterized perspective. Despite its relationship with Closest String, there are almost no "automatic" algorithmic or hardness implications. Minimax Approval Voting is already well-studied the perspective of approximation [9, 18], and is known to admit a PTAS [8]. We focus on the parameterized complexity of Minimax Approval Voting. Our results are summarized in Table 1, and include the following.

- Minimax Approval Voting, when parameterized by d and m is unlikely to admit a polynomial kernel, even though it is trivially FPT even when parameterized only by m (by trying all candidate committees in $\mathcal{O}(2^m)$ time).
- Minimax Approval Voting, when parameterized by d alone, is FPT and admits an algorithm with $\mathcal{O}^\star(d^d)$ running time. On the other hand, when parameterized by k alone, the problem is W[2]-hard. [1]
- Minimax Approval Voting, when parameterized by n, is FPT, however, it is unlikely to have a polynomial kernel even when parameterized by n and k. This is an adaptation of the proofs in [3]. Also, Minimax Approval Voting admits a randomized algorithm with running time $\mathcal{O}^\star(2^{kn}c^k)$.
- Minimax Approval Voting with Outliers is W[1]-hard even when parameterized by s, d and n.
- Minimax Approval Voting with Outliers, the version of the problem where we seek to minimize the number of outliers, is unlikely to admit a PTAS unless P = NP. An adaptation of our proof implies that Closest to Most Strings is also unlikely to have a PTAS unless P = NP, strengthening a previous hardness result [5].

Summary of Results for Minimax Approval Voting		
Parameter	Kernel	FPT
d	No [Theorem 2]	Yes [Theorem 4]
d, m	No [Theorem 2]	Yes [Trivial]
k	N/A	No [Theorem 3]
n	No [Theorem 3]	Yes [ILP]
n, k	No [Theorem 3]	Yes [Theorem 6]

2. PRELIMINARIES

We work in the social choice setting where there are n *voters* and m *candidates*. We let $\mathcal{V} = \{v_1, \ldots, v_n\}$ denote the set of all voters and $\mathcal{C} = \{c_1, \ldots, c_m\}$ denote the set of all candidates. We use the notation $[n]$ to refer to the set $\{1, 2, \ldots, n\}$. A *bit vector* is a word over the binary alphabet $\{0, 1\}$. For a bit vector u, the character (or bit) at the i^{th} position is denoted by $u[i]$. The *weight* of a bit vector u is defined to be the number of ones in the word u.

Let U be a finite set, and $\{u_1, \ldots, u_m\}$ be an arbitrary but fixed ordering of the elements of U. If S is a subset of U, we use $\overline{S}$ to denote the characteristic vector of S, which is a word of length $|U|$ over the binary alphabet $\{0, 1\}$, with a 1 in the i^{th} position if and only if $u_i \in S$. Similarly, if s is a bit vector, we use $\mathcal{J}(s)$ to denote the corresponding set. We sometimes abuse language and refer interchangeably to a set and its characteristic vector. For bit vectors u and v of the same length, we use $d(u, v)$ to denote the Hamming

[1] We use the $\mathcal{O}^\star$ notation to suppress factors that are polynomial in the size of the instance.

distance between u and v, which is the number of positions i where $u[i] \neq v[i]$.

We now define the social choice problem that is central to this work.

> **Minimax Approval Voting**
> Input: A set of alternatives $\mathcal{C} := \{c_1, \ldots, c_m\}$, a collection of votes $\{v_1, \ldots, v_n\}$, where each vote v_i is an element of $\{0,1\}^m$ (or equivalently a subset of $\mathcal{C}$), and positive integers d and k.
> Question: Is there a subset of $\mathcal{X} \subseteq \mathcal{C}$ of size *exactly* k, such that the Hamming distance between $\overline{\mathcal{X}}$ and v_i is at most d for all $1 \leq i \leq n$?

The Minimax Approval Voting problem is closely related to the very well-studied Closest String problem over the binary alphabet, which we describe below for completeness.

> **Closest String**
> Input: A set of n strings $s_1, \ldots, s_n$ over $\{0,1\}$ of length m, and positive integers d and k.
> Question: Is there a string s of length m such that the Hamming distance between s and s_i is at most d for all $1 \leq i \leq n$?

Note that the Minimax Approval Voting problem is exactly the Closest String problem with the additional restraint that the output word has weight k. Note that despite the similarity, there is no trivial computational reduction between the two problems. Informally speaking, the additional constraint can either make the Minimax Approval Voting problem computationally easier or harder. However, we note that any algorithm that enumerates *all* closest strings to a given collection will also solve the Minimax Approval Voting problem.

For $\alpha \subseteq \{m, n, d, k\}$ use the notation $[\alpha]-$Minimax Approval Voting to refer to the Minimax Approval Voting problem when parameterized by the parameters in α. We also introduce the closely related problem of Minimax Approval Voting with Outliers, which is inspired by the analogous and well-studied variant of the Closest String problem, namely Closest to Most Strings.

> **Minimax Approval Voting with Outliers**
> Input: A set of alternatives $\mathcal{C} := \{c_1, \ldots, c_m\}$, a collection of votes $\{v_1, \ldots, v_n\}$, where each vote v_i is an element of $\{0,1\}^m$ (or equivalently a subset of $\mathcal{C}$), and positive integers s, d and k.
> Question: Is there a subset of $\mathcal{X} \subseteq \mathcal{C}$ of size *exactly* k, and a subset $\mathcal{W} \subseteq \mathcal{V}$ of size at least s, such that the Hamming distance between $\overline{\mathcal{X}}$ and w is at most d for all $w \in \mathcal{W}$?

The analogous problem for binary strings is as follows.

> **Closest to Most Strings**
> Input: A set of n strings $s_1, \ldots, s_n$ over $\{0,1\}$ of length m, and positive integers s, d and k.
> Question: Is there a string w of length m for which at least s strings among $s_1, \ldots, s_n$ have Hamming distance at most d from w?

As before, for $\alpha \subseteq \{m, n, d, k, s\}$ we use the notation $[\alpha]-$Minimax Approval Voting with Outliers to refer to the Minimax Approval Voting with Outliers problem when parameterized by the parameters in α. For a special instance where every voter votes for a committee of a fixed size, we use t to denote the weight of each vote. Also, we use $s^\star$ to denote the *dual* parameter in the context of outliers, that is, when we are asking if there is a committee that is at a Hamming distance of at most d from *all but at most* $s^\star$ voters.

Parameterized Complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where k is the parameter. A *kernelization* algorithm is a set of preprocessing rules that runs in polynomial time and reduces the instance size with a guarantee on the output instance size. This notion is formalized below.

Definition 1. **[Kernelization]** *[19, 14] A kernelization algorithm for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(x,k) \in \Gamma^* \times \mathbb{N}$, outputs, in time polynomial in $|x| + k$, a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(x,k) \in \Pi$ if and only if $(x',k') \in \Pi$ and (b) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the kernel, and the function g is referred to as the size of the kernel. If $g(k) = k^{\mathcal{O}(1)}$ then we say that Π admits a polynomial kernel.*

For many parameterized problems, it is well established that the existence of a polynomial kernel would imply the collapse of the polynomial hierarchy to the third level (or more precisely, $\mathsf{CoNP} \subseteq \mathsf{NP/Poly}$). Therefore, it is considered unlikely that these problems would admit polynomial-sized kernels. For showing kernel lower bounds, we simply establish reductions from these problems.

Definition 2. **[Polynomial Parameter Transformation]** *[4] Let Γ_1 and Γ_2 be parameterized problems. We say that Γ_1 is polynomial time and parameter reducible to Γ_2, written $\Gamma_1 \leq_{Ptp} \Gamma_2$, if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$, and a polynomial $p : \mathbb{N} \to \mathbb{N}$, and for all $x \in \Sigma^*$ and $k \in \mathbb{N}$, if $f((x,k)) = (x',k')$, then $(x,k) \in \Gamma_1$ if and only if $(x',k') \in \Gamma_2$, and $k' \leq p(k)$. We call f a polynomial parameter transformation (or a PPT) from Γ_1 to Γ_2.*

This notion of a reduction is useful in showing kernel lower bounds because of the following theorem.

Theorem 1. *[4, Theorem 3] Let P and Q be parameterized problems whose derived classical problems are P^c, Q^c, respectively. Let P^c be NP-complete, and $Q^c \in \mathsf{NP}$. Suppose there exists a PPT from P to Q. Then, if Q has a polynomial kernel, then P also has a polynomial kernel.*

3. MINIMAX APPROVAL VOTING

In this section, we outline our results for Minimax Approval Voting. The problem was shown to be NP-hard in [17] using a reduction from the Vertex Cover problem. This motivates the search for fixed-parameter tractable algorithms for Minimax Approval Voting.

Observe that Minimax Approval Voting is easily FPT by exhaustive search when parameterized by m. We now show that it is unlikely to admit a polynomial kernel even when parameterized by d and m. This follows from the proof of the hardness of Closest String in [3], but adapted to ensure that the number of ones in the output string is fixed.

We describe the details of the construction for completeness, but only sketch the proof of equivalence due to space constraints.

THEOREM 2. MINIMAX APPROVAL VOTING *does not admit a polynomial kernel when parameterized by d and m unless* $\mathsf{CoNP} \subseteq \mathsf{NP/Poly}$.

PROOF. We prove the statement through a PPT reduction from CNF-SAT parameterized by the number of variables, adapting the ideas used in [3]. Given a CNF-SAT formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_q$ with variables $x_1, x_2, \ldots, x_p$, we obtain an instance of MINIMAX APPROVAL VOTING as follows. We begin by transforming F to F' with $2p$ variables, such that each clause has length p or 1, where p is the number of variables in F. To do this, we add p new variables $y_1, y_2, \ldots, y_p$. First, we add p new clauses to the formula. These new clauses have length 1 and are negations of the new variables. Next, for every clause C_i that has less than p variables, we replace it with C_i' where $C_i' = C_i \vee y_1 \vee y_2 \vee \ldots \vee y_{p-k}$ where $|C_i| = k$. So the transformed formula is given by:

$$F' = C_1' \wedge C_2' \wedge \ldots \wedge C_q' \wedge \neg y_1 \wedge \neg y_2 \wedge \ldots \wedge \neg y_p$$

To see that F is satisfiable if and only if F' is satisfiable, note that the satisfying assignment to F can be extended to a satisfying assignment of F' by setting all the y_i's to 0. Conversely, a satisfying assignment for F' must set all the y's to 0 to satisfy all the singleton clauses. The remaining clauses C_i' must each be satisfied by one of the original variables x_i, and so this assignment also satisfies the original clauses C_i. We will refer to each singleton clause $\neg y_i$ as C_{q+i}'.

We will now obtain an instance of MINIMAX APPROVAL VOTING from F'. The instance will have a total of $q+13p-8$ strings, each of length $6p-4$. The string set S consists of two types of strings. The first set S_1 will contain a string for each of the clauses in F', thus consisting of $q+p$ strings. The second set S_2 will contain 4 strings for each $i \in [3p-2]$, thus accounting for $12p-8$ strings. For each variable x_j (or y_j) and a clause C_i' we define a two bit string as follows.

$$X_{i,j}(\text{or } Y_{i,j}) = \begin{cases} 01 & \text{if } C_j \text{ contains } x_i \\ 10 & \text{if } C_j \text{ contains } \bar{x}_i \\ 00 & \text{otherwise} \end{cases}$$

- For every clause C_i' with $1 \leq i \leq q$, we add a string s_i to S_1, where $s_i = X_{i,1}X_{i,2}\ldots X_{i,p}Y_{i,1}Y_{i,2}\ldots Y_{i,p}\{10\}^{p-2}$.
- For every clause C_{q+i}' with $1 \leq i \leq p$, we add a string s_{q+i} to S_1, where $s_i = \{00\}^{p+i-1}10\{00\}^{2p-2-i}$.
- We add the following four strings to S_2 $\forall i \in [3p-2]$.

$$\begin{aligned} a_i &= \{00\}^{i-1}11\{00\}^{3p-2-i} \\ b_i &= \{00\}^{i-1}00\{00\}^{3p-2-i} \\ c_i &= \{11\}^{i-1}11\{11\}^{3p-2-i} \\ d_i &= \{11\}^{i-1}00\{11\}^{3p-2-i} \end{aligned}$$

Thus we get an instance of $(\mathcal{C}, \mathcal{V}, k, d)$MINIMAX APPROVAL VOTING by setting $\mathcal{V} = S_1 \uplus S_2$, the number of candidates (or the length of the strings) is $m = 6p-4$, the number of voters (or strings) is $n = q + 13p - 8$, and the maximum Hamming distance is $d = 3p-2$, where p and q are the number of variables and clauses respectively of the original CNF-SAT instance F.

The forward direction of the equivalence is established by translating an assignment to a string that is consistent with the construction described above. Towards the reverse direction, we make the following claim about the structure of a valid string in the reduced instance.

CLAIM 1. *If there exists a string s such that $d(s,v) \leq 3p-2 \forall v \in S_2$, then $s[2i] \neq s[2i-1] \forall i \in [3p-2]$.*

PROOF. For any $i \in [3p-2]$ we look at the four strings corresponding to it in S_2. Then we look at two strings of length $6p-6$, which are $\{00\}^{i-1}\{00\}^{3p-2-i}$ and $\{11\}^{i-1}\{11\}^{3p-2-i}$. The first is subsequence of a_i and b_i, and the second is subsequence of c_i and d_i. These subsequences are also complements of each other, hence any string s has to be at a distance of $3p-3$ from at least one of them. If s has a distance at least $3p-3$ with the first, then it is at a distance at least $3n-3$ with a_i and b_i, else it is at a distance at least $3p-3$ with c_i and d_i. Now, (a_i, b_i) and (c_i, d_i) differ at only two positions, $2i-1$ and $2i$. In these two positions, one of the two strings has 00 while the other has 11, so if a string is to be at a distance of $3p-2$ from both of them, it must have 10 or 01 in these two positions. Otherwise it will differ at both positions with one of the strings, which in addition to the existing Hamming distance of $3p-3$ will result in a total distance of $3p-1$, which is a contradiction. So, if $d(s,v) \leq 3p-2 \forall v \in S_2$, then $s[2i] \neq s[2i-1] \forall i \in [3p-2]$. Let us call such a string (i.e. a string that belongs to $\{01, 10\}^{6p-4}$) a 'well-formed' string. $\square$

Thus, if the reduced instance admits a solution, then it clearly corresponds to a satisfying assignment. For arguing the reverse direction of the equivalence, it remains to be shown that this assignment is indeed satisfying. This is easily checked, and the details are deferred to a full version due to lack of space.

LEMMA 1. *$(\mathcal{C}, \mathcal{V}, k, d)$ is a YES-instance of* MINIMAX APPROVAL VOTING *if and only if F' is satisfiable.*

PROOF. For the forward direction, let f be a satisfying assignment for F'. We will now produce a string s that is at a distance of at most $3p-2$ from all the strings in S. Let t_z be a two bit string for each variable z where $z = x_i$ or y_i for $i \in [p]$. We define t_z as 10 if z is set to 0 and 01 otherwise. Set $s = t_{x_1}t_{x_2}\ldots t_{x_p}t_{y_1}t_{y_2}\ldots t_{y_p}\{10\}^{p-2}$. It is clear that s is a well-formed string, and so will have a distance of exactly $3p-2$ from any string in S_2. Every string in S_1 is an encoded clause. For the first q clauses, each C_j' has exactly p variables. By the definition of s, the clause string s_j matches with s in the trailing $2p-4$ locations. In the remaining positions, consider the p variables that do not appear in C_j'. These correspond to the 00's in the string, and produce a Hamming distance of exactly p from s at these $2p$ positions. Now of the remaining $2p$ locations, s must match s_j in at least two places, precisely at the variable that satisfies the clause C_j'. In the worst case, it differs at the remaining $2p-2$ places, and we get a total Hamming distance $d(s, s_j) \leq 3p-2$. For the singleton clauses, C_j' contains only one variable and s must match the string s_j at the positions corresponding to that variable. In the remaining $6p-6$ positions, s_j differs from s at exactly $3p-3$ positions. So the Hamming distance is $d(s, s_j) = 3p-3 \leq 3p-2$. So

s is at the appropriate Hamming distance from all strings in S_1 is a valid output for the MINIMAX APPROVAL VOTING instance.

In the backward direction, we note that the output string s of the MINIMAX APPROVAL VOTING instance must be a well-formed string, as shown by Claim 1. To obtain a satisfying assignment for F' from s, we set x_i to 0 if $s[2i-1]s[2i] = 10$ and to 1 otherwise. Similarly, we set y_i to 0 if $s[2n+2i-1]s[2n+2i] = 10$ and to 1 otherwise. All the variables get unique assignments as string s is well-formed. To prove that this is a satisfying assignment, we must show that each clause contains at least one variable that the assignment sets to 1. Consider the string s_j corresponding to a clause C'_j in S_1. For the first q clauses, this clause contains p variables. For each of the variables that C'_j does not contain, s differs from s_j in one position, resulting in a partial Hamming distance of p. For the remaining positions, the bits of s_j are set to 01 or 10. Since s is a well-formed string, it cannot differ with exactly one bit out of the two bits corresponding to each of the variables. We claim that s must match both positions corresponding to a variable for at least one variable in C'_j. If not, then s differs from s_j at two positions for each of the p variables, resulting in a partial Hamming distance of $2p$, which in addition to the distance incurred earlier, results in a total distance of $3p$, which is a contradiction. So one variable must match at both positions. This variable satisfies the clause C'_j. For the singleton clauses, s will differ from s_j in exactly half of the $6p-6$ positions that are filled with 00 in s_j while s is well-formed. The only remaining places are the positions where s_j has 10. If s contains 01 at these positions, the total Hamming distance will become $3p-3+2 = 3p-1$ which is a contradiction, as s is to be a distance of $3p-2$ or less from all the strings of S. So s must contain 10 at those positions, which correspond to the variable y_j. The variable is set to 0 and this satisfies the clause. Thus s provides a valid satisfying assignment for F'. □

We have proved that $(\mathcal{C}, \mathcal{V}, k, d)$ is a YES instance of MINIMAX APPROVAL VOTING if and only if F' is satisfiable. We know that F' is satisfiable if and only if F is. Thus we have obtained an instance of MINIMAX APPROVAL VOTING from an instance of CNF-SAT, where the parameters m and d are polynomial in the number of variables in the original CNF-SAT instance. The procedure is carried out in polynomial time, and so this is a Polynomial Time Transformation from CNF-SAT parameterized by the number of variables to MINIMAX APPROVAL VOTING. We also know the following.

It is well-known that CNF-SAT does not admit a polynomial kernel when parameterized by the number of variables unless $\mathsf{CoNP} \subseteq \mathsf{NP/Poly}$. Consequently, combined with Lemma 1, we prove that MINIMAX APPROVAL VOTING does not admit a polynomial kernel unless $\mathsf{CoNP} \subseteq \mathsf{NP/Poly}$. This concludes the proof.

We note that MINIMAX APPROVAL VOTING is FPT when parameterized by the number of votes – the ILP approach used by [15] can be easily extended to accommodate the committee size constraint. However, we show that the problem is unlikely to admit a polynomial kernel even when parameterized by the number of votes and k.

THEOREM 3. *MINIMAX APPROVAL VOTING, parameterized by the number of votes n and k, does not admit a polynomial kernel unless $\mathsf{CoNP} \subseteq \mathsf{NP/Poly}$.*

PROOF. We show a *polynomial parametric transformation* from HITTING SET parameterized by the number of sets to MINIMAX APPROVAL VOTING. Since [11] shows kernelization hardness for HITTING SET, this rules out polynomial kernels for MINIMAX APPROVAL VOTING as well.

Consider a HITTING SET instance $(U, \mathcal{F}, k')$, where U is the universe of elements, $\mathcal{F}$ a family of subsets of U, $n' = |\mathcal{F}|$ and $m' = |U|$. Without loss of generality, assume that every set $S_i \in \mathcal{F}$ is of the same size l' – we will later show that this (seemingly) restricted version of the problem is equivalent to the original HITTING SET problem. We reduce this instance to a MINIMAX APPROVAL VOTING instance $(\mathcal{C}, \mathcal{V}, k, d)$, where the number of candidates $m = m'$, the number of voters $n = n'$, committee size $k = k'$ and the maximum permitted Hamming distance $d = k' + l' - 1$

Let $U = \{1 \ldots n\}$ and let $\mathcal{F} = \{S_1, \ldots S_m\}$ For each set $S_i \in \mathcal{F}$, let $v_i = \bar{S_i}$, the characteristic vector of S_i. Let $\mathcal{V} = \{v_1 \ldots v_n\}$ be our vote set.

CLAIM 2. *If $(U, \mathcal{F}, k')$ is a YES-instance for HITTING SET, then $(\mathcal{C}, \mathcal{V}, k, d)$ is a YES-instance for MINIMAX APPROVAL VOTING.*

PROOF. Let $S \subseteq U$ be a valid set of size at most k for $\mathcal{F}$, and let v be the indicator vector of S. Clearly, v has exactly k 1s. Also, each vote v_i has exactly l' 1s, at least one out of which overlaps with a 1 in v. Thus, $d(s, s_i) \leq k' + l' - 1 = d$ for every s_i. In other words, $(\mathcal{C}, \mathcal{V}, k, d)$ is a YES-instance for MINIMAX APPROVAL VOTING with v being a valid consensus. □

CLAIM 3. *If $(\mathcal{C}, \mathcal{V}, k, d)$ is a YES-instance for MINIMAX APPROVAL VOTING then $(U, \mathcal{F}, k')$ is a YES-instance for HITTING SET.*

PROOF. Let v be a consensus string for the MINIMAX APPROVAL VOTING instance $(\mathcal{C}, \mathcal{V}, k, d)$. We show that the corresponding set $S = \mathcal{J}(v)$, whose indicator vector is v, is a valid hitting set for the HITTING SET instance $(U, \mathcal{F}, k')$.

First of all, observe that $|S| = k = k'$, since v has exactly k 1s. Also, $d(v, v_i) \leq d = k' + l' - 1$ for every v_i, which means that some 1 in v overlaps with at least one 1 in each v_i. Rephrasing in HITTING SET terminology, S hits at least one element of every S_i, i.e. $(U, \mathcal{F}, k')$ is a YES-instance for HITTING SET, with S being a valid hitting set. □

To complete our proof, we also need to show that the general HITTING SET problem is equivalent to a restricted case where each set of $\mathcal{F}$ is of size exactly l. We call this version of the problem l-REGULAR HITTING SET, and show the following.

CLAIM 4. *Every instance of HITTING SET can be turned into an equivalent instance of REGULAR HITTING SET.*

PROOF. Let S_i be the largest set in $\mathcal{F}$, and let $l = |S_i|$. for every other set S_j with size l', add $l - l'$ dummy elements to S_j and to U. Let the modified instance be $(U', \mathcal{F}')$.

Clearly, a hitting set of size k for $(U, \mathcal{F})$ will indeed be a hitting set for $(U', \mathcal{F}')$. Conversely, consider a hitting set S' for $(U', \mathcal{F}')$. If S' does not contain any of the dummy elements, then S' is a hitting set for $(U, \mathcal{F})$ as well. On the other hand, any dummy element in S' will hit only one set from $\mathcal{F}'$, and hence can be replaced by any other element from that set. Thus, any solution for $(U', \mathcal{F}')$ can be transformed into an equivalent solution for $(U, \mathcal{F})$. □

We note that the reduction above also establishes the $W[2]$-hardness of the problem when parameterized by k alone.

We finally turn to two FPT algorithms. The first one is an algorithm when parameterized by d alone (extending the approach of [15]). The second algorithm considers the combined parameter n and k. The first algorithm uses a depth-bounded branching strategy, while the second one uses the method of color coding.

Algorithm 1: Recursive Procedure **MAVd**(v, δ)

input : Candidate string v and integer δ
Global variables: Set of voters $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, integer d

output: A string v^* with $\max_{i \in [n]} d(v^*, v_i) \leq d$ and $d(v^*, v) \leq \delta$ if it exists, and 'not found' otherwise.

if $\delta < 0$ return NOT FOUND;
if $d(v, v_i) > d + \delta$ for some $i \in [n]$ return NOT FOUND;
if $d(v, v_i) \leq d$ for all $i \in [n]$ return v;
for *some* $i \in [n]$ *such that* $d(v, v_i) > d$*:* **do**
    $P_1 = \{p \mid v[p] = 1, v_i[p] = 0\}$;
    $P_2 = \{q \mid v[q] = 0, v_i[q] = 1\}$;
    for *all* $p \in P_1$ **do**
        for *all* $q \in P_2$ **do**
            $v' = v$;
            $v'[p] = 0$;
            $v'[q] = 1$;
            v_{ret} =`MAVd` $(v', \delta - 2)$;
            If $v_{ret} \neq$NOT FOUND then return v_{ret};
return NOT FOUND;

We first discuss the FPT algorithm for the parameter d. The algorithm starts with some suitable string v having k 1's as the 'candidate string'. If there is some string v_i with $i \in [n]$ that differs from v at more than d positions, then we attempt to bring the candidate string 'closer' to v_i. We do this by removing some selected member of the committee that the voter corresponding to v_i did not vote for, and replacing him with another member that v_i did vote for, thus maintaining the strength of the committee at k. This means we change one of the k 1's in v to a zero, at a position p where $v_i[p] = 0$, and change one of the 0's in v to a one, at a position q where $v_i[q] = 1$. As in the approach used in [15], our algorithm stops either if the candidate string has moved too 'far away' from the initial string, or if it finds a solution. The size of the search tree for the recursion can be limited to $\mathcal{O}(d^d)$, as shown.

Theorem 4. *Given a set of strings $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and an integer d, Algorithm 1 determines in time $\mathcal{O}^\star(d^d)$ whether there is a string v such that $\max_{i \in [n]} d(v, v_i) \leq d$ and computes such a v if it exists.*

Proof. **Running time.** The parameter δ is initialized to d and is decremented by 2 in each step of recursion. The recursion stops when $\delta < 0$. So the depth of the search tree is at most $d/2$. In a single step of recursion, the algorithm selects a string v_i such that $d(v, v_i) > d$. It creates a new subcase for each pair of positions from P_1 and P_2 where v_i differs from v. As $|P_1| + |P_2| = d + 1$, this results in a branching of at most $((d + 1)/2)^2$. Thus the tree size is bounded from above by $((d+1)/2)^{2 \times \frac{d}{2}}$ or $\mathcal{O}(d^d)$. Every step of the recursion requires time that is polynomial in n and d, so the total running time is $\mathcal{O}^\star(d^d)$.

Correctness. We show that Algorithm 1 finds a string v such that $\max_{i \in [n]} d(v, v_i) \leq d$ if it exists. We explicitly show the correctness of only the first step of recursion; the correctness of the algorithm follows by inductive application of the same argument. For the initial candidate string, consider an arbitrary string from $\mathcal{V}$. Without loss of generality, we select v_1. Note that v_1 must contain at least $k - d$ 1's, otherwise it cannot be at a distance of less than $d + 1$ from any string that contains k 1's. If v_1 contains more than k 1's, we use the first k of these to create a candidate string v that adopts the first k 1's of v_1 and places 0 at all other positions. If v_1 contains less than k 1's, then we adopt the first $k - d$ 1's into v and add d more 1's to v at arbitrary locations. This gives us our initial candidate string.

In the situation that v satisfies $\max_{i \in [n]} d(v, v_i) \leq d$ for all $i \in [n]$, we immediately find the solution, i.e. v. If not, then there must exist some v_i such that $d(v, v_i) > d$. For the branching, we consider the positions where v and v_i differ, i.e. $P_1 = \{p \mid v[p] = 1, v_i[p] = 0\}$ and $P_2 = \{q \mid v[q] = 0, v_i[q] = 1\}$. The algorithm successively creates subcases for every pair of positions $p \in P_1$ and $q \in P_2$, and creates a new candidate by altering v to v' so that $v'[p] = 0$ and $v'[q] = 1$. Such a move is correct if the size of the committee, i.e. the number of 1's in v' remains k and the move brings the candidate string 'closer' to v^*, the solution string. It is clear that the number of 1's in the candidate string is always constant at k. We must show that at least one of the subcases is a correct move. We know that v^* differs from v_i in at most d locations. So, for all pairs p and q where $p + q = d + 1$, at least one pair must try a pair of positions that bring the candidate string closer to v^*.

Lemma 5 shows that it is correct to omit those branches where the candidate string v satisfies $d(v, v_i) > d + \delta$ for some $i \in [n]$.

Claim 5. *If there are two strings $v_i, v_j \in \mathcal{V}$ such that $d(v_i, v_j) > 2d$, then there is no string v such that $\max_{i \in [n]} d(v, v_i) \leq d$.*

Proof. Hamming distance follows triangle inequality. So if given that $d(v_i, v_j) > 2d$, then $d(v_i, v) + d(v, v_j) > 2d$ for every v. Thus either $d(v_i, v) > d$ or $d(v, v_j) > d$ (or both). $\square$

This completes the proof for Theorem 4. We now turn to a randomized algorithm parameterized by n and k. This based on the classic Color Coding approach introduced in [1].

Theorem 5. Minimax Approval Voting *admits a randomized* FPT *algorithm, parameterized by the number of voters n and the committee size k.*

Proof. Let $(\mathcal{C}, \mathcal{V} := \{v_1, \ldots, v_n\}, k, d)$ be an instance of Minimax Approval Voting. We call a subset $\mathcal{X} \subseteq \mathcal{C}$ a *consensus committee* if $\mathcal{X}$ is a valid Minimax Approval Voting solution; in other words, the weight of $\mathcal{X}$ is k and further,$d(\overline{\mathcal{X}}, v) \leq d$ for all $v \in \mathcal{V}$.

We call a mapping $\phi : \mathcal{C} \to [k]$ a *k-coloring* of the candidate set $\mathcal{C}$. Note that a coloring partitions $\mathcal{C}$ into k color classes, $\mathcal{C}_1 \ldots \mathcal{C}_k$. A coloring ϕ is a *good coloring* if there exists a consensus committee $\mathcal{X}$ which picks exactly one candidate of each color. Further, we call a consensus committee

$\mathcal{X}$ to be *nice* to a vote v_i with respect to a color j if $\mathcal{X}$ contains some element of $\mathcal{J}(v_i) \cap \mathcal{C}_j$. We define $\omega(\mathcal{X}, v_i)$ to be the following k-length characteristic vector:

$$\omega(\mathcal{X}, v_i)[j] = \begin{cases} 1 & \text{if } \mathcal{X} \text{ is nice to } v_i \text{ on color } j, \\ 0 & \text{otherwise,} \end{cases}$$

and we refer to this as the *niceness vector* of v_i with respect to $\mathcal{X}$.

The algorithm. Assume that we have a good coloring ϕ. For every vote v_i, we guess a niceness vector ω_i. Given such a guess, our task now is to determine if there exists a consensus committee $\mathcal{Y}$ that respects all of these vectors, that is, if $\omega_i[j] = 1$, then $\mathcal{Y}$ picks some candidate in $\mathcal{J}(v_i) \cap C_j$. This, however, is easily checked as follows. For every color j, let $\mathcal{V}_j$ be the set of votes v_i for which $\omega_i[j] = 1$. Note that $\mathcal{Y}$ must pick *one* candidate from $\mathcal{C}_j$ that intersects the sets $\mathcal{J}(v_i) \cap \mathcal{C}_j$ for every $i \in \mathcal{V}_j$. If the family:

$$\{\mathcal{J}(v_i) \cap \mathcal{C}_j \mid i \in \mathcal{V}_j\}$$

is an intersecting family, then we pick any element in the common intersection; otherwise it is clear that we must reject this guess as there is no $\mathcal{Y}$ that can intersect all sets while only picking one element from $\mathcal{C}_j$.

By repeating this procedure for all possible guesses for collections of nice vectors, we ensure that we will find a valid consensus committee whenever there exists one.

Correctness of the algorithm. Assume that there exists a consensus committee $\mathcal{X}$ of size k. We try sufficiently many different random colorings to ensure that we find a coloring that assigns each member of $\mathcal{X}$ a unique color.

Now, consider a good coloring ϕ and a consensus set $\mathcal{X}$. For a vote v_i, let $N_i = \{\phi(c_i) \mid c_i \in \mathcal{X} \cap \mathcal{J}(v_i)\}$, i.e. N_i is the set of all colors that $\mathcal{X}$ is nice on for the vote v_i. Our algorithm explores all possible choices of N_i – in particular, the algorithm cannot miss C_i induced by a valid consensus committee. Given the right collection of niceness vectors, our algorithm finds a consensus committee that respects all of them if one exists, so while the output of the algorithm may differ from $\mathcal{X}$, it is an equally valid choice of a consensus committee.

Running time. We start by guessing a random coloring ϕ for $\mathcal{C}$. By standard arguments, we will find a good coloring with high probability if we try $\mathcal{O}(e^k)$ different colorings.

Further, we need to guess what colors are nice for each votes. To get the nice colors right for one vote, this may take up to 2^k guesses in the worst case. Over all the votes, this adds an $(2^k)^n = 2^{kn}$ factor to the running time.

Determining whether a valid consensus exists for a given guess can be done in $\mathcal{O}(mnk)$ time. Thus, the overall running time of the algorithm is $\mathcal{O}(e^k \cdot 2^{kn} \cdot mnk) = c^{kn} \cdot m^{\mathcal{O}(1)}$ for a suitable choice of c. $\square$

THEOREM 6. MINIMAX APPROVAL VOTING *is in* FPT, *parameterized by the number of voters n and the committee size k.*

4. MAV WITH OUTLIERS

In this section, we show the hardness of approximation of the MINIMAX APPROVAL VOTING WITH OUTLIERS problem, and also establish that it is W[2]-hard when parameterized by s, d and k. In [5], the authors show a randomized reduction from MAX-2-SAT to CLOSEST TO MOST STRINGS, and used the result in [16] to show that for some $\epsilon > 0$ there is no polynomial time $(1 + \epsilon)$-approximation algorithm for CLOSEST TO MOST STRINGS unless P=NP. In this section, we adapt their reduction, using a tweak to fix the number of ones in the output, and a slightly different set of "fixing strings", replacing the randomized engine with a deterministic one. We now describe the details of our approach.

THEOREM 7. *For some $\epsilon > 0$, if there is a polynomial time $(1 + \epsilon)$-approximation algorithm for* MINIMAX APPROVAL VOTING WITH OUTLIERS*, then* P=NP.

PROOF. We give a deterministic reduction from MAX-2-SAT to MINIMAX APPROVAL VOTING WITH OUTLIERS with a fixed k number of 1's in the output. As input, we take an instance of MAX-2-SAT comprised of q clauses $C_1, C_2, \ldots, C_q$ and p variables $x_1, x_2, \ldots, x_p$, where each clause is a disjunction of two literals appearing as either x_i or $\bar{x_i}$ for some $i \in [p]$, and r which is the number of clauses to be satisfied. The output of the reduction will be an instance of MINIMAX APPROVAL VOTING WITH OUTLIERS with a string set S consisting of $q + 2p(q - r + 1)$ strings of length $2p$. Let $l = q - r + 1$. Here the $2pl$ strings are 'fixing' strings intended to force a structure in the solutions, while the first q strings represent an encoding of each clause as follows. For every clause C_j containing the variables from $x_1, x_2, \ldots, x_p$, the corresponding string $s_j = s_j(1)s_j(2)\ldots s_j(2p)$, where:

$$s_j(2i-1)s_j(2i) = \begin{cases} 01 & \text{if } C_j \text{ contains } x_i \\ 10 & \text{if } C_j \text{ contains } \bar{x_i} \\ 00 & \text{otherwise} \end{cases}$$

The fixing strings shall be of the form $\{00, 11\}^p$, or 'double strings'. There are l identical copies of a single 'block' of fixing strings. A block B_t is defined as follows. For every $i \in [p]$, we add two strings to the block.

$$\begin{aligned} a_i^t &= \{00\}^{i-1} 11 \{00\}^{p-i} \\ b_i^t &= \{11\}^{i-1} 00 \{11\}^{p-i} \\ B_t &= \textstyle\bigcup_{i \in [p]} \{a_i^t, b_i^t\} \end{aligned}$$

Thus every block B_t consists of $2p$ strings of length $2p$. All $l = q - r + 1$ copies of the block together with the string encoding of each clause comprise the strings for our instance of MINIMAX APPROVAL VOTING WITH OUTLIERS, i.e. $S = B_1 \cup B_2 \cup \ldots B_l \cup \{s_1, s_2, \ldots, s_q\}$. So $\mid S \mid = n = q + 2p(q - r + 1)$ and the length of each string is $m = 2p$. The distance parameter is set $d = p$ and the number of strings that need to satify the constraint is $s = r + 2pl$ (i.e. the maximum number of outliers is $q - r$). The number of 1's in the output string is $k = p$.

For the forward direction, assume there exists an assignment ϕ to the variables $x_1, x_2, \ldots, x_p$ that satisfies r clauses. We encode this assignment in a string $\bar{\phi}$ of length $2p$ as follows. $\bar{\phi} = \bar{\phi}(1)\bar{\phi}(2)\ldots\bar{\phi}(2p)$, where:

$$\bar{\phi}(2i-1)\bar{\phi}(2i) = \begin{cases} 01 & \text{if } x_i \text{ is set to True} \\ 10 & \text{if } x_i \text{ is set to False} \end{cases}$$

Thus the assignment string $\bar{\phi}$ belongs to $\{01, 10\}^p$, i.e. it is 'well-formed' and has p 1's. The Hamming distance $d(\bar{\phi}, w)$ for any string w where w is a double string is exactly p, so $d(\bar{\phi}, w) \le p$ for every $w \in B_1 \cup \ldots \cup B_l$. So the fixing strings are not outliers. Now, for every clause C_j where $j \in [q]$, the string s_j contains exactly $2p-4$ 0's for the $p-2$ variables that do not appear in C_j. This produces a Hamming distance of $p-2$ from the well-formed $\bar{\phi}$. Of the two variables that do appear in C_j, at least one must be set to true (or false if it appears negatively) in the assignment ϕ if C_j is satisfied by ϕ. The string locations for this variable must match exactly with its encoding in $\bar{\phi}$. So the Hamming distance caused by the variable that do appear in C_j cannot exceed 2. So for a clause C_j that is satisfied by ϕ, the distance $d(\bar{\phi}, s_j) \leq (p-2)+2 = p$. Thus the satisfied clauses do not produce outliers. Since ϕ satisfies at least r clauses, there can be at most $q-r$ outliers, which satisfies the conditions of Minimax Approval Voting with Outliers.

For the backward direction, let ψ be a string that satisfies $d(\psi, w) \leq n$ for $w \in S$ with a maximum of $q-r$ outliers and has exactly p 1's. We first show that ψ must necessarily be a well-formed string.

Claim 6. *ψ belongs to $\{01, 10\}^p$.*

Proof. Assume to the contrary that $\exists i$ such that $\psi(2i-1)\psi(2i) = 00$ or 11.

- If $\psi(2i-1)\psi(2i) = 00$, consider the string $a_i^t = \{00\}^{i-1}11\{00\}^{p-i}$. At the locations $2i-1$ and $2i$, the Hamming distance is exactly 2. In the remaining locations of ψ there are exactly p 1's, which cause a further Hamming distance of p. The total distance $d(\psi, a_i^t) = p+2$. Thus a_i^t is an outlier. However, S contains $l = q-r+1$ copies of a_i^t in the blocks $B_1, B_2, \ldots, B_l$. This is a contradiction, as there can be at most $q-r$ outliers.
- If $\psi(2i-1)\psi(2i) = 11$, consider the string $b_i^t = \{11\}^{i-1}00\{11\}^{p-i}$. Again, there is a Hamming distance of 2 at the locations $2i-1$ and $2i$. The remaining length of ψ contains exactly $p-2$ 1's and $2p-2-(p-2) = p$ 0's, which cause a further Hamming distance of p. Thus the total distance is $p+2$ and b_i^t is an outlier. However, there are $l = q-r+1$ copies of b_i^t in S. This is a contradiction.

So ψ must be a well-formed string, and none of the fixing strings are the outliers. ☐

There can be a maximum of $q-r$ outliers in the q remaining strings, so there must be at least r strings satisfying $d(\psi, p) \leq p$. For these r clause-encoding strings, a Hamming distance of exactly $p-2$ is caused by the $2p-4$ locations corresponding to variables not appearing in the clause. In the locations corresponding to the variables that do appear, the string contains 01 or 10. Note that ψ is well-formed, so the Hamming distance caused by these locations can be either 2 or 0 for each variable. If both variables cause a distance of 2, then total distance will be $p+2$ and the string will not satisfy $d(\psi, p) \leq p$. So at least one variable location produces a distance of 0, i.e. it matches with ψ. So if ψ is used as an assignment vector, setting x_i =True if $\psi(2i-1)\psi(2i) = 01$ and x_i =False if $\psi(2i-1)\psi(2i) = 10$, then as such clauses will be satisfied. As there are at least r such clauses, the assignment corresponding to ψ satisfies the conditions for max-2-sat.

This completes the polynomial time reduction from max-2-sat to Minimax Approval Voting with Outliers. If there exists an $\epsilon > 0$ such that there is a polynomial time $(1+\epsilon)$-approximation algorithm for Minimax Approval Voting with Outliers, then this would also give an approximation for max-2-sat. However, it has been shown in [16] that it is NP-hard to compute a a 22/21-approximately optimal solution for max-2-sat. So for some suitable $\epsilon > 0$, Minimax Approval Voting with Outliers cannot have a $(1+\epsilon)$-approximation algorithm unless P=NP.

Theorem 8. *For some $\epsilon > 0$, if there is a polynomial time $(1+\epsilon)$-approximation algorithm for* Closest to Most Strings*, then* P=NP.

Proof. In this reduction from max-2-sat to Closest to Most Strings we use a similar construct as in Theorem 7, but must ensure that the reasoning is valid even for the case where the output string of the reduced instance does not have exactly $k = p$ 1's. To accommodate this possibility, we include two new fixing strings in every block B_t of double strings. These new strings are simply:

$$c_t = \{00\}^p \; d_t = \{11\}^p$$

Thus every block B_t now contains $2p+2$ strings of length $2p$ and the total number of strings in S is $q + (2p+2)l$. The remaining parameters retain their values, so maximum outliers is $q-r$ and the required Hamming distance from each string is $d = p$.

For the forward direction of the reduction, note that both c_t and d_t for each value of $t \in [l]$ are double strings. So the encoded assignment string $\bar{\phi}$, which is a well-formed string, will have a Hamming distance of exactly p from all copies of the new fixing strings. The remainder of the argument is identical to that of the forward direction for Theorem 7.

For the backward direction, let ψ be the output string of the Closest to Most Strings instance. Note that we cannot yet use Claim 6 to show that ψ is well-formed as that proof used that fact that the string contained exactly p 1's.

Claim 7. *ψ contains exactly p 1's.*

Proof. Assume to the contrary that ψ contains either $> p$ or $< p$ 1's. Then:

- If ψ contains $> p$ 1's, then consider the string $c_t = \{00\}^p$. The Hamming distance $d(c_t, \psi)$ is exactly equal to the number of 1's in ψ, which is greater than the distance parameter $d = p$. So c_t is an outlier. However, if c_t is an outlier, then every one of the $l = q-r+1$ copies of c_t in S is an outlier. But the ψ cannot produce more than $q-r$ outliers. So this is a contradiction.
- If ψ contain $< p$ 1's, then consider the string $d_t = \{11\}^p$. Since a mismatch occurs at every location of ψ except those which contain 1, the Hamming distance $d(c_t, \psi)$ is exactly $2p - v$ where v is the number of 1's in ψ. But v is less than p, so $2p - v > p$. Thus d_t is an outlier. However, there are $l = q-r+1$ copies of d_t in S, and these must all then be outliers. But the maximum number of outliers is $q-r$, so this is a contradiction.

Thus, ψ must contain exactly p 1's. ☐

The rest of the argument follows identically from that of Theorem 7. Thus there exists some $\epsilon > 0$ such that Closest to Most Strings does not have a $(1+\epsilon)$-approximation algorithm, unless P=NP.

Note that both of the previous reductions have involved using string duplicates in the string set S. It is also possible to reduce an instance of MAX-2-SAT to an equivalent instance of CLOSEST TO MOST STRINGS which does not use duplicates, thus proving a stronger result.

THEOREM 9. *For some $\epsilon > 0$, if there is a polynomial time $(1+\epsilon)$-approximation algorithm for* CLOSEST TO MOST STRINGS (WITHOUT DUPLICATES)*, then* P=NP.

PROOF. Once again, we use the construction used in Theorem 8, with some modifications to ensure that no two strings in S are identical. To do this, we insert a suitable addendum at the end of every string in S. For the clause-encoded strings, we insert $l = q - r + 1$ 0's at the end to create new strings of length $2p + l$, with the final l locations forming the padding. For every clause C_j:

$$s'_j = s_j\{0\}^t$$

For the fixing strings, we add a unique addendum (or 'padding') at the end of the strings of each block. Thus the fixing strings are now defined as follows.

$$\begin{aligned} a_i^t &= \{00\}^{i-1}11\{00\}^{p-i}\{0\}^{t-1}1\{0\}^{l-t} \\ b_i^t &= \{11\}^{i-1}00\{11\}^{p-i}\{0\}^{t-1}1\{0\}^{l-t} \\ c_t &= \{00\}^{p}\{0\}^{t-1}1\{0\}^{l-t} \\ d_t &= \{11\}^{p}\{0\}^{t-1}1\{0\}^{l-t} \\ B_t &= c_t \cup d_t \cup \textstyle\bigcup_{i\in[p]} \{a_i^t, b_i^t\} \end{aligned}$$

With these addendums, every block B_t consists of distinct strings of length $2p+l$. All l blocks together with the string encoding of each clause comprise the string set, i.e. $S = B_1 \cup B_2 \cup \ldots B_l \cup \{s'_1, s'_2, \ldots, s'_q\}$. So $\mid S \mid = n = q + 2pl$ and the length of each string is $m = 2p + l$. The distance parameter is changed to $d = p+1$ and the number of strings that need to satify the constraint remains $s = r + 2pl$ (i.e. the maximum number of outliers is $q - r$). There is no constraint on the number of 1's in the output.

For the forward direction, we against construct an assignment string $\bar{\phi}$ as before, and this time add a padding of l 0's to the end, to create a string of length $2p + l$. Note that the initial segment of $\bar{\phi}$ is still well-formed, so it results in a partial Hamming distance of exactly p from the initial segment of each of the fixing strings, which is always a double string. In addition, the padded segment now adds a distance of 1, corresponding to the location of the padded 1 in the fixing string. Thus the total Hamming distance of $\bar{\phi}$ from each of the fixing strings is $p + 1$, and these strings are not outliers. Next, consider the clause-encoded strings. These strings have a padding segment that is identical to that of $\bar{\phi}$, and does not contribute to the Hamming distance. So, using the same argument as in Theorem 7, at least r of these strings are not outliers. Thus $\bar{\phi}$ is closest to at least $r + 2pl$ strings of S.

For the backward direction, let ψ be the output string of the reduced instance. We now show that ψ must necessarily have exactly p 1's.

CLAIM 8. *ψ contains exactly p 1's, all contained in the first $2p$ locations of ψ.*

PROOF. Let p_1 be the number of 1's in the first segment of the first $2p$ locations of ψ, and p_2 be the number of 1's in the last l locations or the padded segment. So the total 1's is given by $p_1 + p_2$.

- Consider the string $c_t = \{00\}^p\{0\}^{t-1}1\{0\}^{l-t}$. The Hamming distance caused by the first segment is exactly p_1, while the distance caused by the padded segment of the padded segment is at most $p_2 + 1$. Thus the total distance is $p_1 + p_2 + 1$. Since ψ cannot be at a distance of more than $p + 1$ from every c_t for $t \in [l]$, we have $p_1 + p_2 + 1 \le p + 1$. So $p_1 + p_2 \le p$.
- Consider the string $d_t = \{11\}^p\{0\}^{t-1}1\{0\}^{l-t}$. The Hamming distance caused by the first segment is exactly $2p - p_1$, while the distance caused by the padded segment is again at most $p_2 + 1$. But ψ cannot be at a distance of more than $p + 1$ from every d_t for $t \in [l]$, so we have $2p - p_1 + p_2 + 1 \le p + 1$. So $p \le p_1 + p_2$.

But p_1 and p_2 are non-negative, so the only solution to the two inequalities is $p_1 = p$ and $p_2 = 0$. Thus, ψ must contain exactly p 1's, all in the first segment, and no 1's in the padded segment. ☐

The remaining argument follows from the backward direction of Theorem 7. Thus there exists some $\epsilon > 0$ such that CLOSEST TO MOST STRINGS WITHOUT DUPLICATES does not have a $(1 + \epsilon)$-approximation algorithm, unless P=NP.

THEOREM 10. MINIMAX APPROVAL VOTING WITH OUTLIERS *is* W[1]-hard*, even when parameterized by s, d and k.*

PROOF. We show a reduction from the k-CLIQUE problem, parameterized by k.

Starting from a graph $G = (V, E)$. We construct an election instance $\mathcal{E} = (\mathcal{C}, \mathcal{V}, s, k, d)$, such that $\mathcal{E}$ has a k-consensus if and only if G has a clique of size k. We set s, the number of voters to satisfy, as $s = \binom{k}{2}$ and $d = k - 2$.

- For each vertex v in the graph, add a candidate c_v.
- Each vote is a bit string of length $|V|$, where each bit corresponds to a vertex. For each edge (u_1, u_2) in the graph, add a vote v_e which sets $v[u_1] = v[u_2] = 1$ and $s[i] = 0$ everywhere else.

Suppose that G has a clique C of size k, and let x be the characteristic vector of the vertices in C. Note that the weight of x is k. A clique of size k contains $\binom{k}{2}$ edges. Further, for every edge e contained in C, both endpoints of e are contained in C – thus, $d(x, v_e) = k - 2$. Therefore, x is a valid consensus vote for $\mathcal{E}$ and $\mathcal{E}$ is a YES-instance.

Conversely, assume that $\mathcal{E}$ is a YES-instance and let x be a valid consensus committee for $\mathcal{E}$. We show that $C = \mathcal{J}(x)$ will induce a clique on the graph G.

Indeed, $\mathcal{V}$ contains $\binom{k}{2}$ votes, with Hamming distance d to x. These votes correspond to edges in G – there are $\binom{k}{2}$ edges within C, hence C forms a clique of size k.

CLIQUE parameterized by the clique size k is a well-known W[1]-hard problem [12]. The above reduction bounds all the three parameters, s, d, k in terms of the clique size k in the original instance – it follows that MINIMAX APPROVAL VOTING WITH OUTLIERS is W[1]-hard even when parameterized by s, d and k together. ☐

5. ACKNOWLEDGMENTS

The first author is supported by the INSPIRE Faculty Scheme, DST India (project DSTO-1209). This work was carried out when the third author visited the Indian Institute of Science. His visit was sponsored by the INSPIRE fellowship of the first author.

REFERENCES

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

[2] Haris Aziz, Serge Gaspers, Joachim Gudmundsson, Simon Mackenzie, Nicholas Mattei, and Toby Walsh. Computational aspects of multi-winner approval voting. In *Proc. of the International Conference On Autonomous Agents & Multiagent Systems* (AAMAS), 2015 *To Appear.*

[3] Manu Basavaraju, Fahad Panolan, Ashutosh Rai, M.S. Ramanujan, and Saket Saurabh. On the kernelization complexity of string problems. In *Computing and Combinatorics*, volume 8591 of *Lecture Notes in Computer Science*, pages 141–153, 2014.

[4] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel Bounds for Disjoint Cycles and Disjoint Paths. In *Proceedings of the 17th Annual European Symposium,on Algorithms* (ESA), volume 5757 of *Lecture Notes in Computer Science*, pages 635–646, Springer 2009.

[5] Christina Boucher, Gad M. Landau, Avivit Levy, David Pritchard, and Oren Weimann. On approximating string selection problems with outliers. In Theor. Comput. Sci, pages 107–114, 2013.

[6] Steven J. Brams and Peter C. Fishburn. Going from theory to practice: the mixed success of approval voting. *Social Choice and Welfare*, 25(2-3):457–474, Springer 2005.

[7] Robert Bredereck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. In *Tsinghua Science and Technology*, volume 19, pages 358–373. IEEE, 2014.

[8] Jaroslaw Byrka and Krzysztof Sornat. PTAS for minimax approval voting. In *Proc. of Web and Internet Economics - 10th International Conference* (WINE), volume 8877 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2014.

[9] Ioannis Caragiannis, Dimitris Kalaitzis, and Evangelos Markakis. Approximation algorithms and mechanism design for minimax approval voting. In *AAAI*. AAAI Press, 2010.

[10] Vincent Conitzer. Making decisions based on the preferences of multiple agents. *Commun. ACM*, 53(3):84–94, 2010.

[11] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In *Automata, Languages and Programming* (ICALP), pages 378–389. Springer, 2009.

[12] Rodney G Downey and Michael R Fellows. *Fundamentals of Parameterized complexity*. Springer, 2013.

[13] Edith Elkind, Piotr Faliszewski, Piotr Skowron, and Arkadii Slinko. Properties of multiwinner voting rules. In *Proc. of the International Conference On Autonomous Agents & Multiagent Systems* (AAMAS), pages 53–60, IFAAMAS/ACM, 2014.

[14] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume 3. Springer, 2006.

[15] Jens Gramm, Rolf Niedermeier, Peter Rossmanith, et al. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.

[16] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, July 2001.

[17] Rob LeGrand. Analysis of the minimax procedure. Technical report, Technical Report WUCSE-2004-67, Department of Computer Science and Engineering, Washington University, St. Louis, Missouri, 2004.

[18] Rob LeGrand, Evangelos Markakis, and Aranyak Mehta. Some results on approximating the minimax solution in approval voting. In *Proc. of the International Conference On Autonomous Agents & Multiagent Systems* (AAMAS), page 198. IFAAMAS, 2007.

[19] Rolf Niedermeier. Invitation to fixed-parameter algorithms. *Habilitationschrift, University of Tübingen*, 2002.