The Learnability of Voting Rules*

Ariel D. Procaccia† Aviv Zohar‡ Yoni Peleg§ Jeffrey S. Rosenschein¶

Abstract

Scoring rules and voting trees are two broad and concisely-representable classes of voting rules; scoring rules award points to alternatives according to their position in the preferences of the voters, while voting trees are iterative procedures that select an alternative based on pairwise comparisons. In this paper, we investigate the PAC-learnability of these classes of rules. We demonstrate that the class of scoring rules, as functions from preferences into alternatives, is efficiently learnable in the PAC model. With respect to voting trees, while in general a learning algorithm would require an exponential number of samples, we show that if the number of leaves is polynomial in the size of the set of alternatives, then a polynomial training set suffices. We apply these results in an emerging theory: automated design of voting rules by learning.

keywords: Computational social choice, Computational learning theory, Multiagent systems

1 Introduction

Voting is a well-studied method of preference aggregation, in terms of its theoretical properties, as well as its computational aspects [6, 21]; various practical, implemented applications that use voting exist [9, 13, 12].

In an election, n voters express their preferences over a set of m alternatives. To be precise, each voter is assumed to reveal linear preferences—a ranking of the alternatives. The outcome of the election is determined according to a *voting rule*. In this paper we will consider two families of voting rules: scoring rules and voting trees.

Scoring Rules. The predominant—ubiquitous, even—voting rule in real-life elections is the *Plurality* rule. Under Plurality, each voter awards one point to the alternative it ranks first, i.e., its most preferred alternative. The alternative that accumulated the most points, summed over all voters, wins the election. Another example of a voting rule is the *Veto* rule: each voter "vetoes" a single alternative; the alternative that was vetoed by the fewest voters wins the election. Yet a third example is the *Borda* rule: every voter awards $m-1$ points to its top-ranked alternative,

*This paper subsumes two earlier conference papers [23, 24].

†Microsoft Israel R&D Center, 13 Shenkar Street, Herzeliya 46725, Israel, email: arielpro@gmail.com. The author was supported in this work by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

‡School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel, and Microsoft Israel R&D Center, 13 Shenkar Street, Herzeliya 46725, Israel, email: avivz@cs.huji.ac.il.

§School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel, email: jonip@cs.huji.ac.il.

¶School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel, email: jeff@cs.huji.ac.il.

$m-2$ points to its second choice, and so forth—the least preferred alternative is not awarded any points. Once again, the alternative with the most points is elected.

The above-mentioned three voting rules all belong to an important family of voting rules known as *scoring rules*. A scoring rule can be expressed by a vector of parameters $\vec{\alpha} = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$, where each α_l is a real number and $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m$. Each voter awards α_1 points to its most-preferred alternative, α_2 to its second-most-preferred alternative, etc. Predictably, the alternative with the most points wins. Under this unified framework, we can express our three rules as:

- *Plurality*: $\vec{\alpha} = \langle 1, 0, \ldots, 0 \rangle$.
- *Borda*: $\vec{\alpha} = \langle m-1, m-2, \ldots, 0 \rangle$.
- *Veto*: $\vec{\alpha} = \langle 1, \ldots, 1, 0 \rangle$.

A good indication of the importance of scoring rules is given by the fact that they are exactly the family of voting rules that are anonymous (indifferent to the identities of the voters), neutral (indifferent to the identities of the alternatives), and consistent (an alternative that is elected by two separate sets of voters is elected overall) [28].

Voting Trees. Some voting rules rely on the concept of *pairwise elections*: alternative a beats alternative b in the pairwise election between a and b if the majority[1] of voters prefers a to b. Ideally, we would like to select an alternative that beats every other alternative in a pairwise election, but such an alternative (called a *Condorcet winner*) does not always exist.

However, there are other prominent voting rules that rely on the concept of pairwise elections, which select an alternative in a sense "close" to the Condorcet winner. In the Copeland rule, for example, the score of an alternative is the number of alternatives it beats in a pairwise election; the alternative with the highest score wins. In the Maximin rule, the score of an alternative is the number of votes it gets in its worst pairwise election (the least number of voters that prefer it to some alternative), and, predictably, the winner is the alternative that scores highest.

When discussing such voting rules, it is possible to consider a more abstract setting. A *tournament* T over A is a complete binary asymmetric relation over A (that is, for any two alternatives a and b, aTb or bTa, but not both). Clearly, the aforementioned majority relation induces a tournament (a beats b in the pairwise election iff aTb). More generally, this relation can reflect a reality that goes beyond a strict voting scenario. For example, the tournament can represent a basketball league, where aTb if team a is expected to beat team b in a game. We denote the set of all tournaments over A by $\mathcal{T} = \mathcal{T}(A)$.

So, for the moment let us look at (pairwise) voting rules as simply functions $f : \mathcal{T} \to A$. The most prominent class of such functions is the class of *binary voting trees*. Each function in the class is represented by a binary tree, with the leaves labeled by alternatives. At each node, the alternatives at the two children compete; the winner ascends to the node (so if a and b compete and aTb, a ascends). The winner-determination procedure starts at the leaves and proceeds upwards towards the root; the alternative that survives to the root is the winner of the election.

For example, assume that the alternatives are a, b, and c, and bTa, cTb, and aTc. In the tree given in Figure 1, b beats a and is subsequently beaten by c in the right subtree, while a beats c in the left subtree. a and c ultimately compete at the root, making a the winner of the election.

[1] We will assume, for simplicity, an odd number of voters.

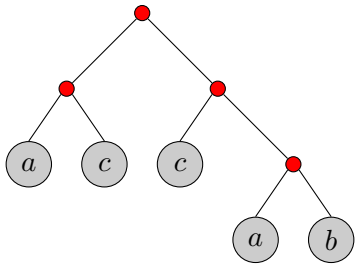

Figure 1: A binary voting tree

Notice that we allow an alternative to appear in multiple leaves; further, some alternatives may not appear at all (so, for example, a singleton tree is a constant function).

Motivation and Setting. We consider the following setting: an entity, which we refer to as the *designer*, has in mind a voting rule (which may reflect the ethics of a society). We assume that the designer is able, for each constellation of voters' preferences with which it is presented, to designate a winning alternative (perhaps with considerable computational effort). In particular, one can think of the designer's representation of the voting rule as a black box that matches preference profiles to winning alternatives. This setting is relevant, for example, when a designer has in mind different properties it wants its rule to satisfy; in this case, given a preference profile, the designer can specify a winning alternative that is compatible with these properties.

We would like to find a concise and easily understandable representation of the voting rule the designer has in mind. We refer to this process as *automated design of voting rules*: given a specification of properties, or, indeed, of societal ethics, find an elegant voting rule that implements the specification. In this paper, we do so by learning from examples. The designer is presented with different preference profiles, drawn according to a fixed distribution. For each profile, the designer answers with the winning alternative. The number of queries presented to the designer must intuitively be as small as possible: the computations the designer has to carry out in order to handle each query might be complex, and communication might be costly.

Now, we further assume that the "target" voting rule the designer has in mind, i.e., the one given as a black box, is known to belong to some family $\mathcal{R}$ of voting rules. We would like to produce a voting rule from $\mathcal{R}$ that is as "close" as possible to the target rule.

By "close," we mean close with respect to the fixed distribution over preference profiles. More precisely, we would like to construct an algorithm that receives pairs of the form (preferences, winner) drawn according to a fixed distribution D over preferences, and outputs a rule from $\mathcal{R}$, such that the probability according to D that our rule and the target rule agree is as high as possible. We wish, in fact, to learn rules from $\mathcal{R}$ in the framework of the formal PAC (Probably Approximately Correct) learning model; a concise introduction to this model is given in Section 2.

In this paper, we look at two options for the choice of $\mathcal{R}$: the family of scoring rules, and the family of voting trees. These are natural choices, since both are broad classes of rules, and both have concise representations. Choosing $\mathcal{R}$ as above, the designer could in principle translate the possibly cumbersome, unknown representation of a voting rule into a succinct one that can be easily understood and computed.

Further justification for our agenda is given by noting that it might be difficult to compute

a voting rule on all instances, but it might be sufficient to simply calculate the election's result on typical instances. The distribution D can be chosen, by the designer, to concentrate on such instances.

Our results. The dimension of a function class is a combinatorial measure of the richness of the class; this dimension is closely related to the number of examples needed to learn the class. We give almost tight bounds on the dimension of the class of scoring rules, providing an upper bound of m, and a lower bound of $m-3$, where m is the number of alternatives in an election. In addition, we show that, given a set of examples, one can efficiently construct a scoring rule that is consistent with the examples, if one exists. Combined, these results imply the following theorem:

Theorem 3.1. *The class of scoring rules over n voters and m alternatives is efficiently learnable for all values of n and m.*

In other words, given a combination of properties that is satisfied by some scoring rule, it is possible to construct a "close" scoring rule in polynomial time.

The situation with respect to the learnability of voting trees is two-fold: in general, due to the expressiveness and possible complexity of binary trees, the number of examples required is exponential in m. However, if we assume that the number of leaves is polynomial in m, then the required number of examples is also polynomial in m. In addition, we investigate the computational complexity of problems associated with the learning process.

It is also worthwhile to ask whether it is possible to extend this approach. Specifically, we pose the question: given a class of voting rules $\mathcal{R}$, if the designer has some general voting rule in mind (rather than a voting rule that is known to belong to $\mathcal{R}$), is it possible to learn a "close" rule from $\mathcal{R}$? We prove, for a natural definition of approximation:

Theorem 5.3. *Let $\mathcal{R}_m^n$ be a family of voting rules of size exponential in n and m. Let $\epsilon, \delta > 0$. For large enough values of n and m, at least a $(1-\delta)$-fraction of the voting rules $f : \mathcal{L}^n \to \{x_1, \ldots, x_m\}$ satisfy the following property: no voting rule in $\mathcal{R}_m^n$ is a $(1/2+\epsilon)$-approximation of f.*

In particular, we show that the theorem holds for scoring rules and small voting trees, thus answering the question posed above in the negative with respect to these classes.

Related Work. Currently there exists a small body of work on learning in economic settings. Kalai [16] explores the learnability (in the PAC model) of rationalizable choice functions. These are functions which, given a set of alternatives, choose the element that is maximal with respect to some linear order. Similarly, PAC learning has very recently been applied to computing utility functions that are rationalizations of given sequences of prices and demands [2].

Another prominent example is the paper by Lahaie and Parkes [17], which considers preference elicitation in combinatorial auctions. The authors show that preference elicitation algorithms can be constructed on the basis of existing learning algorithms. The learning model used, exact learning, differs from ours (PAC learning).

Conitzer and Sandholm [3] have studied automated mechanism design, in the more restricted setting where agents have numerical valuations for different alternatives. They propose automatically designing a truthful mechanism for every preference aggregation setting. However, they find that, under two solution concepts, even determining whether there exists a deterministic mechanism that guarantees a certain social welfare is an $\mathcal{NP}$-complete problem. The authors also show that the problem is tractable when designing a randomized mechanism. In more recent work [5],

Conitzer and Sandholm put forward an efficient algorithm for designing deterministic mechanisms, which works only in very limited scenarios. In short, our setting, goals, and methods are completely different—in the general voting context, even framing computational complexity questions is problematic, since the goal cannot be specified with reference to expected social welfare.

Some authors have studied the computational properties of scoring rules. For instance, Conitzer et al. [6] have investigated the computational complexity of the coalitional manipulation problem in several scoring rules; Procaccia and Rosenschein [21] generalized their results, and finally, Hemaspaandra and Hemaspaandra [14] gave a full characterization. Many other papers deal with the complexity of manipulation and control in elections, and, *inter alia*, discuss scoring rules (see, e.g., [1, 4, 8, 15, 22, 29]).

The computational properties of voting trees have also been investigated. One prominent example is the work of Lang et al. [18], which studied the computational complexity of selecting different types of winners in elections governed by voting trees. Fischer et al. [10] investigated the power of voting trees in approximating the maximum degree in a tournament.

Structure of the Paper. In Section 2 we give an introduction to the PAC model. In Section 3, we present our results with respect to scoring rules. In Section 4, we investigate voting trees. In Section 5, we discuss a possible extension of our approach. We conclude in Section 6.

2 Preliminaries

In this section we give a very short introduction to the PAC model and the generalized dimension of a function class. A more comprehensive (and slightly more formal) overview of the model, and results concerning the dimension, can be found in [20].

In the PAC model, the learner is attempting to learn a function $f : Z \to Y$, which belongs to a class $\mathcal{F}$ of functions from Z to Y. The learner is given a *training set*—a set $\{z_1, z_2, \ldots, z_t\}$ of points in Z, which are sampled i.i.d. (independently and identically distributed) according to a distribution D over the sample space Z. D is unknown, but is fixed throughout the learning process. In this paper, we assume the "realizable" case, where a target function $f^*(z)$ exists, and the given training examples are in fact labeled by the target function: $\{(z_k, f^*(z_k))\}_{k=1}^t$. The *error* of a function $f \in \mathcal{F}$ is defined as

$$\mathrm{err}(f) = \Pr_{z \sim D}[f(z) \neq f^*(z)]. \tag{1}$$

$\epsilon > 0$ is a parameter given to the learner that defines the *accuracy* of the learning process: we would like to achieve $\mathrm{err}(h) \leq \epsilon$. Notice that $\mathrm{err}(f^*) = 0$. The learner is also given a *confidence* parameter $\delta > 0$, that provides an upper bound on the probability that $\mathrm{err}(h) > \epsilon$:

$$\Pr[\mathrm{err}(h) > \epsilon] < \delta. \tag{2}$$

We now formalize the discussion above:

Definition 2.1. [20]

1. A *learning algorithm* L is a function from the set of all training examples to $\mathcal{F}$ with the following property: given $\epsilon, \delta \in (0, 1)$ there exists an integer $s(\epsilon, \delta)$—the *sample complexity*—such that for any distribution D on X, if Z is a sample of size at least s where the samples are drawn i.i.d. according to D, then with probability at least $1 - \delta$ it holds that $\mathrm{err}(L(Z)) \leq \epsilon$.

2. L is an *efficient* learning algorithm if it always runs in time polynomial in $1/\epsilon$, $1/\delta$, and the size of the representations of the target function, of elements in X, and of elements in Y.

3. A function class $\mathcal{F}$ is *(efficiently) PAC-learnable* if there is an (efficient) learning algorithm for $\mathcal{F}$.

The sample complexity of a learning algorithm for $\mathcal{F}$ is closely related to a measure of the class's combinatorial richness known as the generalized dimension.

Definition 2.2. [20] Let $\mathcal{F}$ be a class of functions from Z to Y. We say $\mathcal{F}$ *shatters* $S \subseteq Z$ if there exist two functions $f, g \in \mathcal{F}$ such that

1. For all $z \in S$, $f(z) \neq g(z)$.

2. For all $S_1 \subseteq S$, there exists $h \in \mathcal{F}$ such that for all $z \in S_1$, $h(z) = f(z)$, and for all $z \in S \setminus S_1$, $h(z) = g(z)$.

Definition 2.3. [20] Let $\mathcal{F}$ be a class of functions from a set Z to a set Y. The *generalized dimension* of $\mathcal{F}$, denoted by $D_G(\mathcal{F})$, is the greatest integer d such that there exists a set of cardinality d that is shattered by $\mathcal{F}$.

Lemma 2.4. *[20, Lemma 5.1] Let Z and Y be two finite sets and let $\mathcal{F}$ be a set of total functions from Z to Y. If $d = D_G(F)$, then $2^d \leq |\mathcal{F}|$.*

A function's generalized dimension provides both upper and lower bounds on the sample complexity of algorithms.

Theorem 2.5. *[20, Theorem 5.1] Let $\mathcal{F}$ be a class of functions from Z to Y of generalized dimension d. Let L be an algorithm such that, when given a set of t labeled examples $\{(z_k, f^*(z_k))\}_k$ of some $f^* \in \mathcal{F}$, sampled i.i.d. according to some fixed but unknown distribution over the instance space X, produces an output $f \in \mathcal{F}$ that is consistent with the training set. Then L is an (ϵ, δ)-learning algorithm for $\mathcal{F}$ provided that the sample size obeys:*

$$s \geq \frac{1}{\epsilon}\left((\sigma_1 + \sigma_2 + 3)d \ln 2 + \ln\left(\frac{1}{\delta}\right)\right) \tag{3}$$

where σ_1 and σ_2 are the sizes of the representation of elements in Z and Y, respectively.

Theorem 2.6. *[20, Theorem 5.2] Let $\mathcal{F}$ be a function class of generalized dimension $d \geq 8$. Then any (ϵ, δ)-learning algorithm for $\mathcal{F}$, where $\epsilon \leq 1/8$ and $\delta < 1/4$, must use sample size $s \geq \frac{d}{16\epsilon}$.*

3 Learnability of Scoring Rules

Before diving in, we introduce some notation. Let $N = \{1, 2, \ldots, n\}$ be the set of voters, and let $A = \{x_1, x_2, \ldots, x_m\}$ be the set of *alternatives*; we also denote alternatives by $\{a, b, c, \ldots\}$. Let $\mathcal{L} = \mathcal{L}(A)$ be the set of linear preferences[2] over A; each voter has preferences $\succ^i \in \mathcal{L}$. We denote the *preference profile*, consisting of the voters' preferences, by $\succ^N = \langle \succ^1, \succ^2, \ldots, \succ^n \rangle$. A *voting rule* is a function $f : \mathcal{L}^N \to A$, that maps preference profiles to winning alternatives.

[2] A binary relation which is antisymmetric, transitive, and total.

Let $\vec{\alpha}$ be a vector of m nonnegative real numbers such that $\alpha_l \geq \alpha_{l+1}$ for all $l = 1, \ldots, m-1$. Let $f_{\vec{\alpha}} : \mathcal{L}^N \to C$ be the scoring rule defined by the vector $\vec{\alpha}$, i.e., each voter awards α_l points to the alternative it ranks in the l'th place, and the rule elects the alternative with the most points.

Since several alternatives may have maximal scores in an election, we must adopt some method of tie-breaking. Our method works as follows. Ties are broken in favor of the alternative that was ranked first by more voters; if several alternatives have maximal scores and were ranked first by the same number of voters, the tie is broken in favor of the alternative that was ranked second by more voters; and so on.[3]

Let $\mathcal{S}_m^n$ be the class of scoring rules with n voters and m alternatives. Our goal is to learn, in the PAC model, some target function $f_{\vec{\alpha}^*} \in \mathcal{S}_m^n$. To this end, the learner receives a training set $\{(\succ_k^N, f_{\vec{\alpha}^*}(\succ_k^N)\}_k$, where each $\succ_k^N$ is drawn from a fixed distribution over $\mathcal{L}^N$; let $x_{j_k} = f_{\vec{\alpha}^*}(\succ_k^N)$. For the profile $\succ_k^N$, we denote by $\pi_{j,l}^k$ the number of voters that ranked alternative x_j in place l. Notice that alternative x_j's score under the preference profile $\succ_k^N$ is $\sum_l \pi_{j,l}^k \alpha_l$.

3.1 Efficient Learnability of $\mathcal{S}_m^n$

Our main goal in this section is to prove the following theorem.

Theorem 3.1. *For all $n, m \in \mathbb{N}$, the class $\mathcal{S}_m^n$ is efficiently PAC-learnable.*

By Theorem 2.5, in order to prove Theorem 3.1 it is sufficient to validate the following two claims: 1) that there exists an algorithm which, for any training set, runs in time polynomial in n, m, and the size of the training set, and outputs a scoring rule which is consistent with the training set (assuming one exists); and 2) that the generalized dimension of the class $\mathcal{S}_m^n$ is polynomial in n and m.

Remark 3.2. It is possible to prove Theorem 3.1 by using a transformation between scoring rules and sets of linear threshold functions. Indeed, it is well-known that the VC dimension (the restriction of the generalized dimension to boolean-valued functions) of linear threshold functions over $\mathcal{R}^d$ is $d+1$. In principle, it is possible to transform a scoring rule into a linear threshold function that receives (generally speaking) vectors of rankings of alternatives as input. Given a training set of profiles, we could transform it into a training set of rankings and use a learning algorithm.

However, we are interested in producing an accurate scoring rule according to a distribution D on preference profiles, which represents typical profiles. It is possible to consider a many-to-one mapping between distributions over profiles and distributions over the above-mentioned vectors of rankings. Unfortunately, when this procedure is used, it is nontrivial to guarantee that the learned voting rule succeeds according to the original distribution D. Moreover, this procedure seems to require an increase in sample complexity compared to the analysis given below. Therefore, we proceed with the more "direct" agenda outlined above and detailed below.

It is rather straightforward to construct an efficient algorithm that outputs consistent scoring rules. Given a training set, we must choose the parameters of our scoring rule in a way that, for any example, the score of the designated winner is at least as large as the scores of other alternatives. Moreover, if ties between the winner and a loser would be broken in favor of the loser, then the

[3] In case several alternatives have maximal scores and identical rankings everywhere, break ties arbitrarily—say, in favor of the alternative with the smallest index.

winner's score must be strictly higher than the loser's. Our algorithm, given as Algorithm 1, simply formulates all the constraints as linear inequalities, and solves the resulting linear program. The first part of the algorithm is meant to handle tie-breaking. Recall that $x_{j_k} = f_{\vec{\alpha}^*}(\succ_k^N)$.

Algorithm 1 Given a training set of size s, the algorithm returns a scoring rule which is consistent with the given examples, if one exists.

for $k \leftarrow 1 \dots s$ **do**
  $X_k \leftarrow \emptyset$
  for all $x_j \neq x_{j_k}$ **do** ▷ x_{j_k} is the winner in example k
    $\vec{\pi}^{\Delta} \leftarrow \vec{\pi}^k_{j_k} - \vec{\pi}^k_j$
    $l_0 \leftarrow \min\{l : \ \pi^{\Delta}_l \neq 0\}$
    if $\pi^{\Delta}_{l_0} < 0$ **then** ▷ Ties are broken in favor of x_j
      $X_k \leftarrow X_k \cup \{x_j\}$
    end if
  end for
end for
return a feasible solution $\vec{\alpha}$ to the following linear program:

$$\begin{array}{l} \forall k, \ \forall x_j \in X_k, \sum_l \pi^k_{j_k,l}\alpha_l \geq \sum_l \pi^k_{j,l}\alpha_l + 1 \\ \forall k, \ \forall x_j \notin X_k, \sum_l \pi^k_{j_k,l}\alpha_l \geq \sum_l \pi^k_{j,l}\alpha_l \\ \forall l = 1, \dots, m-1 \quad \alpha_l \geq \alpha_{l+1} \\ \forall l, \ \alpha_l \geq 0 \end{array}$$

A linear program can be solved in time that is polynomial in the number of variables and inequalities [26]; it follows that Algorithm 1's running time is polynomial in n, m, and the size of the training set.

Remark 3.3. Notice that any vector $\vec{\alpha}$ with a "standard" representation, that is with rational coordinates such that both numerator and denominator are integers represented by a polynomial number of bits, can be scaled to an equivalent vector of integers which is also polynomially representable. In this case, the scores are always integral. Thus, instead of using a strict inequality in the LP's first set of constraints, we can use a weak inequality with an additive term of 1.

Remark 3.4. Although the transformation between learning scoring rules and learning linear threshold functions mentioned in Remark 3.2 has some drawbacks as a learning method, we conjecture that results on the computational complexity of learning linear threshold functions can be leveraged to obtain computational efficiency. Indeed, well-known algorithms such as Winnow [19] might suit this purpose.

Remark 3.5. Algorithm 1 can also be used to check, with high probability, if the voting rule the designer has in mind is indeed a scoring rule, as described (in a different context) by Kalai [16] (we omit the details here). This further justifies the setting in which the voting rule the designer has in mind is known to be a scoring rule.

So, it remains to demonstrate that the generalized dimension of $\mathcal{S}^n_m$ is polynomial in n and m. The following lemma shows this.

Lemma 3.6. *The generalized dimension of the class $\mathcal{S}_m^n$ is at most m:*

$$D_G(\mathcal{S}_m^n) \leq m.$$

Proof. According to Definition 2.3, we need to show that any set of cardinality $m+1$ cannot be shattered by $\mathcal{S}_m^n$. Let $S = \{\succ_k^N\}_{k=1}^{m+1}$ be such a set, and let h, g be the two social choice functions that disagree on all preference profiles in S. We shall construct a subset $S_1 \subseteq S$ such that there is no scoring rule $f_{\vec{\alpha}}$ that agrees with h on S_1 and agrees with g on $S \setminus S_1$.

Let us look at the first preference profile from our set, $\succ_1^N$. We shall assume without loss of generality that $h(\succ_1^N) = x_1$, while $g(\succ_1^N) = x_2$, and that in $\succ_1^N$ ties between x_1 and x_2 are broken in favor of x_1. Let $\vec{\alpha}$ be some parameter vector. If we are to have $h(\succ_1^N) = f_{\vec{\alpha}}(\succ_1^N)$, it must hold that

$$\sum_{l=1}^{m} \pi_{1,l}^1 \cdot \alpha_l \geq \sum_{l=1}^{m} \pi_{2,l}^1 \cdot \alpha_l, \tag{4}$$

whereas if we wanted $f_{\vec{\alpha}}$ to agree with g we would want the opposite:

$$\sum_{l=1}^{m} \pi_{1,l}^1 \cdot \alpha_l < \sum_{l=1}^{m} \pi_{2,l}^1 \cdot \alpha_l \tag{5}$$

More generally, we define, with respect to the profile $\succ_k^N$, the vector $\vec{\pi}_\Delta^k$ as the vector whose l'th coordinate is the difference between the number of times the winner under h and the winner under g were ranked in the l'th place:[4]

$$\vec{\pi}_\Delta^k = \vec{\pi}_{h(\succ_k)}^k - \vec{\pi}_{g(\succ_k)}^k. \tag{6}$$

Now we can concisely write necessary conditions for $f_{\vec{\alpha}}$ agreeing on $\succ_k^N$ with h or g, respectively, by writing:[5]

$$\vec{\pi}_\Delta^k \cdot \vec{\alpha} \geq 0 \tag{7}$$

$$\vec{\pi}_\Delta^k \cdot \vec{\alpha} \leq 0 \tag{8}$$

Notice that each vector $\vec{\pi}_\Delta^k$ has exactly m coordinates. Since we have $m+1$ such vectors (corresponding to the $m+1$ profiles in S), there must be a subset of vectors that is linearly dependent. We can therefore express one of the vectors as a linear combination of the others. Without loss of generality, we assume that the first profile's vector can be written as a combination of the others with parameters β_k, not all 0:

$$\vec{\pi}_\Delta^1 = \sum_{k=2}^{m+1} \beta_k \cdot \vec{\pi}_\Delta^k \tag{9}$$

Now, we shall construct our subset S_1 of preference profiles as follows:

$$S_1 = \{k \in \{2, \ldots, m+1\} : \ \beta_k \geq 0\} \tag{10}$$

[4]There is some abuse of notation here; if $h(\succ_k^N) = x_l$ then by $\vec{\pi}_{h(\succ_k)}^k$ we mean $\vec{\pi}_l^k$.

[5]In all profiles except $\succ_1^N$, we are indifferent to the direction in which ties are broken.

Suppose, by way of contradiction, that $f_{\vec{\alpha}}$ agrees with h on $\succ_k^N$ for $k \in S_1$, and with g on the rest. We shall examine the value of $\vec{\pi}_\Delta^1 \cdot \vec{\alpha}$:

$$\vec{\pi}_\Delta^1 \cdot \vec{\alpha} = \sum_{k=2}^{m+1} \beta_k \cdot \vec{\pi}_\Delta^k \cdot \vec{\alpha} = \sum_{k \in S_1} \beta_k \cdot \vec{\pi}_\Delta^k \cdot \vec{\alpha} + \sum_{k \notin S_1 \cup \{1\}} \beta_k \cdot \vec{\pi}_\Delta^k \cdot \vec{\alpha} \geq 0 \tag{11}$$

The last inequality is due to the construction of S_1—whenever β_k is negative, the sign of $\vec{\pi}_\Delta^k \cdot \vec{\alpha}$ is non-positive ($f_{\vec{\alpha}}$ agrees with g), and whenever β_k is positive, the sign of $\vec{\pi}_\Delta^k \cdot \vec{\alpha}$ is non-negative (agreement with h).

Therefore, by equation (5), we have that $f(\succ_1^N) \neq x_2 = g(\succ_1^N)$. However, it holds that $1 \notin S_1$, and we assumed that $f_{\vec{\alpha}}$ agrees with g outside S_1—this is a contradiction. □

Theorem 3.1 is thus proven. The upper bound on the generalized dimension of S_m^n is quite tight: in the next subsection we show a lower bound of $m-3$.

3.2 Lower Bound for the Generalized Dimension of $\mathcal{S}_m^n$

Theorem 2.6 implies that a lower bound on the generalized dimension of a function class is directly connected to the complexity of learning it. In particular, a tight bound on the dimension gives us an almost exact idea of the number of examples required to learn a scoring rule. Therefore, we wish to bound $D_G(\mathcal{S}_m^n)$ from below as well.

Theorem 3.7. *For all $n \geq 4$, $m \geq 4$, $D_G(\mathcal{S}_m^n) \geq m-3$.*

Proof. We shall produce an example set of size $m-3$ which is shattered by $\mathcal{S}_m^n$. Define a preference profile $\succ_l^N$, for $l = 3, \ldots, m-1$, as follows. For all l, the voters $1, \ldots, n-1$ rank alternative x_j in place j, i.e., they vote $x_1 \succ_l^i x_2 \succ_l^i \cdots \succ_l^i x_m$. The preferences $\succ_l^n$ (the preferences of voter n in profile $\succ_l^N$) are defined as follows: alternative x_2 is ranked in place l, alternative x_1 is ranked in place $l+1$; the other alternatives are ranked arbitrarily by voter n. For example, if $m=5$, $n=6$, the preference profile $\succ_3^N$ is:

$\succ_3^1$	$\succ_3^2$	$\succ_3^3$	$\succ_3^4$	$\succ_3^5$	$\succ_3^6$
x_1	x_1	x_1	x_1	x_1	x_3
x_2	x_2	x_2	x_2	x_2	x_4
x_3	x_3	x_3	x_3	x_3	x_2
x_4	x_4	x_4	x_4	x_4	x_1
x_5	x_5	x_5	x_5	x_5	x_5

Lemma 3.8. *For any scoring rule $f_{\vec{\alpha}}$ with $\alpha_1 = \alpha_2 \geq 2\alpha_3$ it holds that:*

$$f_{\vec{\alpha}}(\succ_l^N) = \begin{cases} x_1 & \alpha_l = \alpha_{l+1} \\ x_2 & \alpha_l > \alpha_{l+1} \end{cases}$$

Proof. We shall first verify that x_2 has maximal score. x_2's score is at least $(n-1)\alpha_2 = (n-1)\alpha_1$. Let $j \geq 3$; x_j's score is at most $(n-1)\alpha_3 + \alpha_1$. Thus, the difference is at least $(n-1)(\alpha_1 - \alpha_3) - \alpha_1$. Since $\alpha_1 = \alpha_2 \geq 2\alpha_3$, this is at least $(n-1)(\alpha_1/2) - \alpha_1 > 0$, where the last inequality holds for $n \geq 4$.

Now, under preference profile $\succ_l^N$, x_1's score is $(n-1)\alpha_1+\alpha_{l+1}$ and x_2's score is $(n-1)\alpha_1+\alpha_l$. If $\alpha_l = \alpha_{l+1}$, the two alternatives have identical scores, but x_1 was ranked first by more voters (in fact, by $n-1$ voters), and thus the winner is x_1. If $\alpha_l > \alpha_{l+1}$, then x_2's score is strictly higher—hence in this case x_2 is the winner. □

Armed with Lemma 3.8, we will now prove that the set $\{\succ_l^N\}_{l=3}^{m-1}$ is shattered by $\mathcal{S}_m^n$. Let $\vec{\alpha}^1$ be such that $\alpha_1^1 = \alpha_2^1 \geq 2\alpha_3^1 = 2\alpha_4^1 = \cdots = 2\alpha_m^1$, and $\vec{\alpha}^2$ be such that $\alpha_1^1 = \alpha_2^1 \geq 2\alpha_3^1 > 2\alpha_4^1 > \cdots > 2\alpha_m^1$. By the lemma, for all $l = 3, \ldots, m-1$, $f_{\vec{\alpha}^1}(\succ_l^N) = x_1$, and $f_{\vec{\alpha}^2}(\succ_l^N) = x_2$.

Let $T \subseteq \{3, 4, \ldots, m-1\}$. We must show that there exists $\vec{\alpha}$ such that $f_{\vec{\alpha}}(\succ_l^N) = x_1$ for all $l \in T$, and $f_{\vec{\alpha}}(\succ_l^N) = x_2$ for all $l \notin T$. Indeed, configure the parameters such that $\alpha_1 = \alpha_2 > 2\alpha_3$, and $\alpha_l = \alpha_{l+1}$ iff $l \in T$. The result follows directly from Lemma 3.8. □

4 Learnability of Voting Trees

Recall that we are dealing with a set of *alternatives* $A = \{x_1, \ldots, x_m\}$; as before, we will also denote alternatives by $a, b, c \in A$. A *tournament* is a complete binary irreflexive relation T over A; we denote the set of all possible tournaments by $\mathcal{T} = \mathcal{T}(A)$.

A *binary voting tree* is a binary tree with leaves labeled by alternatives. To determine the winner of the election with respect to a tournament T, one must iteratively select two siblings, label their parent by the winner according to T, and remove the siblings from the tree. This process is repeated until the root is labeled, and its label is the winner of the election.

A preference profile $\succ^N$ of a set of voters N induces a tournament $T \in \mathcal{T}(A)$ as follows: aTb (i.e., *a dominates b*) if and only if a majority of voters prefer a to b. Thus, a voting tree is in particular a voting rule, as defined in Section 3. However, for the purposes of this section it is sufficient to regard voting trees as functions $f : \mathcal{T}(A) \to A$, that is, we will disregard the set of voters and simply consider the dominance relation T on A. We shall hereinafter refer to functions $f : \mathcal{T}(A) \to A$ as *pairwise voting rules*.

Let us therefore denote the class of voting trees over m alternatives by $\mathcal{V}_m$; we emphasize the the class depends only on m. We would like to know what the sample complexity of learning functions in $\mathcal{V}_m$ is. To elaborate a bit, since we think of voting trees as functions from $\mathcal{T}$ to A, the sample space is $\mathcal{T}$.

4.1 Large Voting Trees

In this section, we will show that in general, the answer to the above question is that the complexity is exponential in m. We will prove this by relying on Theorem 2.6; the theorem implies that in order to prove such a claim, it is sufficient to demonstrate that the generalized dimension of $\mathcal{V}_m$ is at least exponential in m. This is the task we presently turn to.

Theorem 4.1. *$D_G(\mathcal{V}_m)$ is exponential in m.*

Proof. Without loss of generality, we let $m = 2k+2$. We will associate every distinct binary vector $v = \langle v_1, \ldots, v_k \rangle \in \{0,1\}^k$ with a distinct example in our set of tournaments $S \subseteq \mathcal{T}$. To prove the theorem, we will show that $\mathcal{V}_m$ shatters this set S of size 2^k.

Let the set of alternatives be:

$$A = \{a, b, x_1^0, x_1^1, x_2^0, x_2^1, \ldots, x_k^0, x_k^1\}.$$

For every vector $v \in \{0,1\}^k$, define a tournament T_v as follows: for $i = 1, \ldots, k$, if $v_i = 0$, we let $x_i^0 T_v b T_v x_i^1$; otherwise, if $v_i = 1$, then $x_i^1 T_v b T_v x_i^0$. In addition, for all tournaments T_v, and all $i = 1, \ldots, k$, $j = 0, 1$, a beats x_i^j, but a loses to b. We denote by S the set of these 2^k tournaments.[6] Let f be the constant function b, i.e., a voting tree which consists of only the node b; let g be the constant function a. We must prove that for every $S_1 \subseteq S$, there is a voting tree such that b wins for every tournament in S_1 (in other words, the tree agrees with f), and a wins for every tournament in $S \setminus S_1$ (the tree agrees with g). Consider the tree in Figure 2, which we refer to as the i'th *2-gadget*.

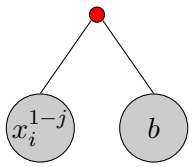

Figure 2: 2-gadget

With respect to this tree, b wins a tournament $T_v \in S$ iff $v_i = j$. Indeed, if $v_i = j$, then $x_i^j T_v b T_v x_i^{1-j}$, and in particular b beats x_i^{1-j}; if $v_i \neq j$, then $x_i^{1-j} T_v b T_v x_i^j$, so b loses to x_i^{1-j}.

Let $v \in \{0,1\}^k$. We will now use the 2-gadget to build a tree where b wins only the tournament $T_v \in S$, and loses every other tournament in S. Consider a balanced tree such that the deepest nodes in the tree are in fact 2-gadgets (as in Figure 3). As before, b wins in the i'th 2-gadget iff $v_i = j$. We will refer to this tree as a v-gadget.

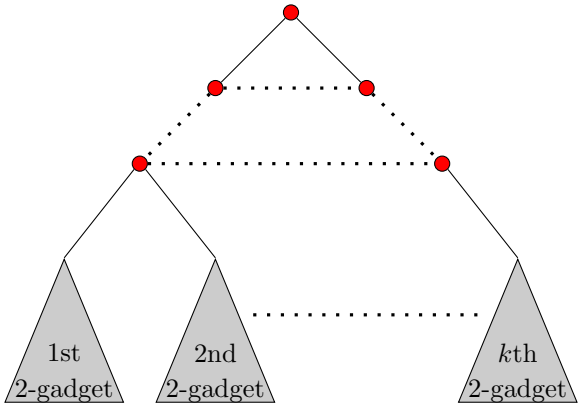

Figure 3: v-gadget

Now, notice that if b wins in each of the 2-gadgets (and this is the case in the tournament T_v), then b is the winner of the entire election. On the other hand, let $v' \neq v$, i.e., there exists $i \in \{1, \ldots, k\}$ such that w.l.o.g. $0 = v'_i \neq v_i = 1$. Then it holds that $x_i^0 T_{v'} b T_{v'} x_i^1$; this implies that x_i^0 wins in the i'th 2-gadget. x_i^0 proceeds to win the entire election, unless it is beaten in some stage by some other alternative x_l^j—but this must be also an alternative that beats b, as it survived the l'th 2-gadget. In any case, b cannot win the election.

[6] The relations described above are not complete, but the way they are completed is of no consequence.

Consider the small extension, in Figure 4, of the v-gadget, which (for lack of a better name) we call the v-gadget*.

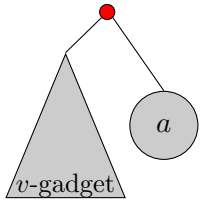

Figure 4: v-gadget*

Recall that, in every tournament in S, a beats any alternative x_j^i but loses to b. Therefore, by our discussion regarding the v-gadget, b wins the election described by the v-gadget* only in the tournament T^v; for any other tournament in S, alternative a wins the election.

We now present a tree and prove that it is as required, i.e., in any tournament in S_1, b is the winner, and in any tournament in $S \setminus S_1$, a prevails. Let us enumerate the tournaments in S_1:

$$S_1 = \{T_{v_1}, \ldots, T_{v_r}\}.$$

We construct a balanced tree, as in Figure 5, where the bottom levels consist of the v_l-gadgets*, for $l = 1, \ldots, r$.

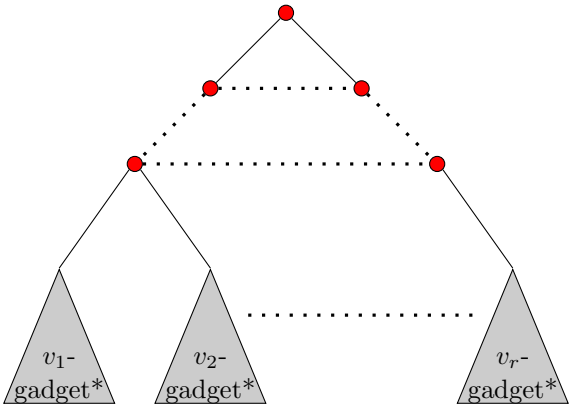

Figure 5: The constructed tree

Let $T_{v_l} \in S_1$. What is the result of this tournament in the election described by this tree? First, note that b prevails in the v_l-gadget*. The only alternatives that can reach any level above the gadgets are a and b, and b always beats a. Therefore, b proceeds to win the election. Conversely, let $T_v \in S \setminus S_1$. Then a survives in *every* v_l-gadget*, for $l = 1, \ldots, r$. a surely proceeds to win the entire election.

We have shown that $\mathcal{V}_m$ shatters S, thus completing the proof. □

Remark 4.2. Even if we restrict our attention to the class of balanced voting trees (corresponding to a playoff schedule), the dimension of the class is still exponential in m. Indeed, any unbalanced

tree can be transformed to an identical (as a voting rule) balanced tree. If the tree's height is h, this can be done by replacing every leaf at depth $d < h$, labeled by an alternative a, by a balanced subtree of height $d - h$ in which all the leaves are labeled by a. This implies that the class of balanced trees can shatter any sample which is shattered by $\mathcal{V}_m$.

Remark 4.3. The proof we have just completed, along with Lemma 2.4, imply that the number of different pairwise voting rules that can be represented by trees is double exponential in m, which highlights the high expressiveness of voting trees.

Theorem 4.1, coupled with Theorem 2.6, implies that the sample complexity of learning arbitrary voting trees is exponential in n and m.

4.2 Small Voting Trees

In the previous section, we have seen that in general, a large number of examples is needed in order to learn voting trees in the PAC model. This result relied on the number of leaves in the trees being exponential in the number of alternatives. However, in many realistic settings one can expect the voting tree to be compactly represented, and in particular one can usually expect the number of leaves to be at most polynomial in m. Let us denote by $\mathcal{V}_m^{(k)}$ the class of voting trees over m alternatives, with at most k leaves. Our goal in this section is to prove the following theorem.

Theorem 4.4. $D_G\left(\mathcal{V}_m^{(k)}\right) = \mathcal{O}(k \log m)$.

This theorem implies, in particular, that if the number of leaves k is polynomial in m, then the dimension of $\mathcal{V}_m^{(k)}$ is polynomial in m. In turn, this implies by Lemma 2.5 that the sample complexity of $\mathcal{V}_m^{(k)}$ is only polynomial in m. In other words, there is a polynomial $p(m, 1/\epsilon, 1/\delta)$ such that, given a training set of size $p(m, 1/\epsilon, 1/\delta)$, any algorithm that returns some tree consistent with the training set is an (ϵ, δ)-learning algorithm for $\mathcal{V}_m^{(k)}$.

To prove the theorem, we require the following straightforward lemma.

Lemma 4.5. $|\mathcal{V}_m^{(k)}| \leq k \cdot m^k \cdot C_{k-1}$, *where* C_k *is the* k*'th Catalan number, given by*

$$C_k = \frac{1}{k+1}\binom{2k}{k}.$$

Proof. The number of voting trees with exactly k leaves is at most the number of binary tree structures multiplied by the number of possible assignments of alternatives to leaves. The number of assignments is clearly bounded by m^k. Moreover, it is well known that the number of rooted ordered binary trees with k leaves is the $(k-1)$ Catalan number. So, the total number of voting trees with exactly k leaves is bounded by $m^k \cdot C_{k-1}$, and the number of voting trees with *at most* k leaves is at most $k \cdot m^k \cdot C_{k-1}$. □

We are now ready to prove Theorem 4.4.

Proof of Theorem 4.4. By Lemma 4.5, we have that

$$|\mathcal{V}_m^{(k)}| \leq k \cdot m^k \cdot C_{k-1}.$$

Therefore, by Lemma 2.4:

$$D_G(\mathcal{V}_m^{(k)}) \leq \log |\mathcal{V}_m^{(k)}| = \mathcal{O}(k \log m).$$

□

4.3 Computational Complexity

In the previous section, we restricted our attention to voting trees where the number of leaves is polynomial in k. We have demonstrated that the dimension of this class is polynomial in m, which implies that the sample complexity of the class is polynomial in m. Therefore, any algorithm that is consistent with a training set of polynomial size is a suitable learning algorithm (Theorem 2.5).

It seems that the significant bottleneck, especially in the setting of automated voting rule design (finding a compact representation for a voting rule that the designer has in mind), is the number of queries posed to the designer, so in this regard we are satisfied that realistic voting trees are learnable. Nonetheless, in some contexts we may also be interested in computational complexity: given a training set of polynomial size, how computationally hard is it to find a voting tree which is consistent with the training set?

In this section we explore the above question. We will assume hereinafter that the structure of the voting tree is known *a priori*. This is an assumption that we did not make before, but observe that, at least for balanced trees, Theorems 4.1 and 4.4 hold regardless. We shall try to determine how hard it is to find an assignment to the leaves which is consistent with the training set. We will refer to the computational problem as Tree-SAT (pun intended).

Definition 4.6. In the Tree-SAT problem, we are given a binary tree, where some of the leaves are already labeled by alternatives, and a training set that consists of pairs (T_j,x_{i_j}), where $T_j \in \mathcal{T}$ and $x_{i_j} \in A$. We are asked whether there exists an assignment of alternatives to the rest of the leaves which is consistent with the training set, i.e., for all j, the winner in T_j with respect to the tree is x_{i_j}.

Notice that in our formulation of the problem, some of the leaves are already labeled. However, it is reasonable to expect any efficient algorithm that finds a consistent tree, given that one exists, to be able to solve the Tree-SAT problem. Hence, an $\mathcal{NP}$-hardness result implies that such an algorithm is not likely to exist.

Theorem 4.7. Tree-Sat *is $\mathcal{NP}$-complete.*

Proof. It is obvious that Tree-SAT is in $\mathcal{NP}$. In order to show $\mathcal{NP}$-hardness, we present a reduction from 3SAT. In this problem, one is given a conjunction of clauses, where each clause is a disjunction of three literals. One is asked whether the given formula has a satisfying assignment. It is known that 3SAT is $\mathcal{NP}$-complete [11].

Given an instance of 3SAT with variables $\{x_1,\ldots,x_m\}$, and clauses $\{l_1^j \vee l_2^j \vee l_3^j\}_{j=1}^k$, we construct an instance of Tree-Sat as follows: the set of alternatives is

$$A = \{a, b, x_1, \neg x_1, c_1, x_2, \neg x_2, c_2, \ldots, x_m, \neg x_m, c_m\}.$$

For each clause j, we define a tournament T_j as some tournament that satisfies the following restrictions:

1. l_1^j, l_2^j and l_3^j beat any other alternative among the alternatives $x_i, \neg x_i$, possibly excluding their own negations.
2. a loses to l_1^j, l_2^j and l_3^j, but beats any other alternative among the alternatives $x_i, \neg x_i$.

In addition, all tournaments in our instance of Tree-SAT satisfy the following conditions:

1. b beats any alternative which corresponds to a literal, but loses to a.

2. For all $i = 1, \ldots, m$, $\neg x_i$ beats x_i.

3. c_i loses to x_i and $\neg x_i$, and beats any other literal and the alternatives a and b. The tournaments are arbitrarily defined with respect to competitions between c_i and c_k, $i \neq k$.

Finally, for each tournament, we require the winner to be alternative b. We now proceed to construct the given (partially assigned) tree. We start, as in the proof of Theorem 4.1, by defining a gadget which we call an *i-gadget*, illustrated in Figure 6.

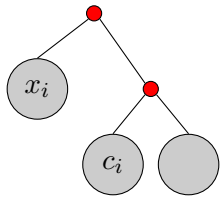

Figure 6: i-gadget.

In this subtree, two leaves are already assigned with x_i and c_i. Now, with respect to any of the tournaments we defined, if we assign $\neg x_i$ to the last leaf, then $\neg x_i$ proceeds to beat c_i, and subsequently beats x_i. If we assign x_i to the third leaf, then x_i beats c_i and wins the election. If we assign any other alternative that is not c_k for some $k = 1, \ldots, m$, then that alternative is defeated by c_i, which in turn is beaten by x_i. Finally, if c_k is assigned, it either loses to c_i and then x_i is the winner, or it beats c_i and proceeds to beat x_i. To conclude the point, either x_i, $\neg x_i$, or c_k for some $k \neq i$ survive the i-gadget.

Using the i-gadgets, we design a tree that will complete the construction of our Tree-SAT instance; the tree is described in Figure 7.

We now prove that this is indeed a reduction. We first have to show that if the given 3SAT instance is satisfiable, there is an assignment to the leaves of our tree (in particular, choices of x_i or $\neg x_i$) such that, for each of the m tournaments, the winner is b. Consider some satisfying assignment to the 3SAT instance. For every literal l_i that is assigned a truth value, we assign the label l_i to the unlabeled leaf of the i-gadget, i.e., we make l_i survive the i-gadget. Now, consider some tournament T_j. At least one of the literals l_1^j, l_2^j or l_3^j must be true; as these three literals beat all other literals in the tournament T_j, one of these three literals reaches the competition versus a, and wins; subsequently, this literal loses to alternative b. Therefore, b is the winner of the election. Since this is true for any $j = 1, \ldots, m$, we have that the assignment is consistent with the given tournaments.

In the other direction, consider an instance of 3SAT which is not satisfiable, and fix some assignment to the leaves of the tree. A first case that we consider is that under this assignment, c_k survives some i-gadget, $i \neq k$. c_k cannot be beaten on the way to the root of the tree, except by another c alternative. Hence, b does not win in any of the constructed tournaments.

A second case to consider is that for each i-gadget, either x_i or $\neg x_i$ survives. The corresponding assignment to the 3SAT instance is not satisfying. Therefore, there is some j such that l_1^j, l_2^j, and l_3^j are all false. This implies that in T_j some other literal other than these three reaches the top of the tree to compete against a, and loses. Subsequently, a competes against b and wins, making a

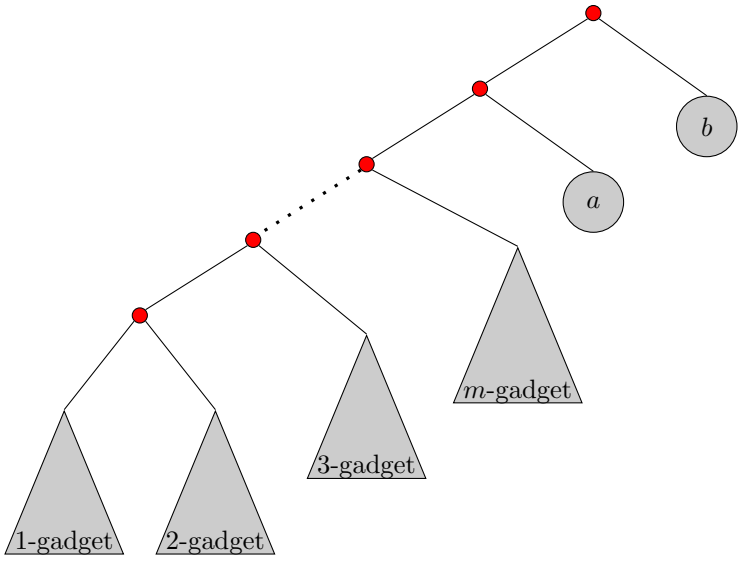

Figure 7: The reduction.

the winner of the election with respect to tournament T_j. Hence, this is not an assignment which is consistent with all tournaments—but this is true with respect to any such assignment. □

Despite Theorem 4.7, it seems that in practice, solving the Tree-Sat problem is sometimes possible; we shall empirically demonstrate this.

Our simulations were carried out as follows. Given a fixed tree structure, we randomly assigned alternatives (out of a pool of 32 alternatives) to the leaves of the tree. We then used this tree to determine the winners in 20 random tournaments over our 32 alternatives. Next, we measured the time it took to find some assignment to the leaves of the tree (not necessarily the original one) which is consistent with the training set of 20 tournaments. We repeated this procedure 10 times for each number of leaves in $\{4, 8, 16, 32, 64\}$, and took the average of all ten runs.

The problem of finding a consistent tree can easily be represented as a constraint satisfaction problem, or in particular as a SAT problem. Indeed, for every node, one simply has to add one constraint per tournament which involves the node and its two children. To find a satisfying assignment, we used the SAT solver zChaff. The simulations were carried out on a PC with a Pentium D (dual core) CPU, running Linux, with 2GB of RAM and a 2.8GHz clock speed.

We experimented with two different tree structures. The first is seemingly the simplest—a binary tree which is as close to a chain as possible, i.e., every node is either a leaf, or the parent of a leaf; we refer to these trees as *caterpillars*. The second is intuitively the most complicated: a balanced tree. Notice that, given that the number of leaves is k, the number of nodes in both cases is $2k - 1$. The simulation results are shown in Figure 8.

In the case of balanced trees, it is indeed hard to find a consistent tree. Adding more sample tournaments would add even more constraints and make the task harder. However, in most elections the number of alternatives is usually not above several dozen, and the problem may still be solvable. Furthermore, the problem is far easier with respect to caterpillars (even though the reduction in Theorem 4.7 builds trees that are "almost caterpillars"). Therefore, we surmise that for many

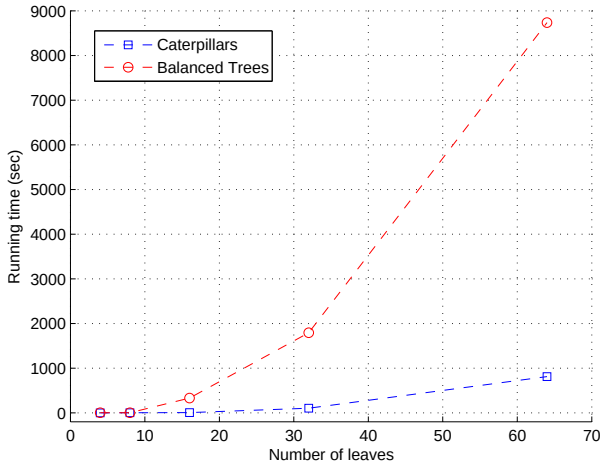

Figure 8: Time to find a satisfying assignment

tree structures, it may be practically possible (in terms of the computational effort) to find a consistent assignment, even when the input is relatively large, while for others the problem is quite computationally hard even in practice.

5 On Learning Voting Rules "Close" to Target Rules

Heretofore, we have concentrated on learning voting rules that are known to be either scoring rules or voting trees. In particular, we have assumed that there is a scoring rule or a voting tree that is consistent with the given training set.

In this section, we push the envelope by asking the following question: given examples that are consistent with some general voting rule, is it possible to learn a scoring rule or a small voting tree that is "close" to the target rule?

Mathematically we are actually asking whether there exist target voting rules f^* such that $\min_{f_{\vec{\alpha}} \in \mathcal{S}_m^n} \mathrm{err}(f_{\vec{\alpha}})$, or $\min_{f \in \mathcal{V}_m^*} \mathrm{err}(f)$, is large. This of course depends on the underlying distribution D. In the rest of this section, the implicit assumption is that D is the simplest nontrivial distribution over profiles, namely the uniform distribution. Nevertheless, the uniform distribution usually does not reflect real preferences of voters; this is an assumption we are making for the sake of analysis. In light of this discussion, the definition of distance between voting rules is going to be the fraction of preference profiles on which the two rules disagree.

Definition 5.1. A voting rule $f : \mathcal{L}^N \to A$ is a *c-approximation* of a voting rule g iff f and g agree on a c-fraction of the possible preference profiles:

$$|\{\succ^N \in \mathcal{L}^N : \ f(\succ^N) = g(\succ^N)\}| \geq c \cdot (m!)^n.$$

In other words, the question is: given a training set $\{(\succ_k^N, f(\succ_j^N)\}_k$, where $f : \mathcal{L}^N \to A$ is some voting rule, how hard is it to learn a scoring rule or a voting tree that c-approximates f, for c that is close to 1?

It turns out that the answer is: it is impossible. We shall first give an extreme example for the case of scoring rules. Indeed, there are voting rules that disagree with any scoring rule on almost all of the preference profiles; if the target rule f is such a rule, it is impossible to find, and of course impossible to learn, a scoring rule that is "close" to f.

In order to see this, consider the following voting rule that we call *flipped veto*: each voter awards one point to the alternative it ranks *last*; the winner is the alternative with the most points. In addition, ties are broken according to the lexicographic order on alternative names. This rule is of course not reasonable as a preference aggregation method, but still—it is a valid voting rule.

Proposition 5.2. *Let $f_{\vec{\alpha}}$ be a scoring rule that is a c-approximation of flipped veto. Then $c \leq 1/m$.*

Proof. Let $\succ^N$ be a preference profile such that $f_{\vec{\alpha}}(\succ^N) = \text{flipped veto}(\succ^N) = x^*$, for some $x^* \in A$. Define a set $B_{\succ^N} \subseteq \mathcal{L}^N$ as follows: each profile in the set is obtained by switching the place of an alternative $x \in A$, $x \neq x^*$, with the place of x^*, in the ordering of each voter that did not rank x^* last.[7] For a preference profile $\succ_1^N \in B_{\succ^N}$ that was obtained by switching x with x^*, it holds that the winner under flipped veto is x^*, since its score did not decrease as a result of the switches, while its situation in terms of tie-breaking remained the same (that is, its name did not change). In addition, under $f_{\vec{\alpha}}$ the situation of x in $\succ_1^N$, with respect to score and tie-breaking, is at least as good as the situation of x^* in $\succ^N$ (voters that have not switched the two alternatives are ones that rank x^* last, and the score of the other alternatives remains unchanged). Note that it might be the case that x and x^* have the same score under $\succ_1^N$; however, since flipped veto$(\succ^N) = x^*$ it holds that x^* is ranked last by at least one voter in $\succ^N$, and hence in $f_{\vec{\alpha}}(\succ_1^N)$ ties between x and x^* are broken in favor of x. It follows that $f_{\vec{\alpha}}(\succ_1^N) = x$. Therefore, for any preference profile in $B_{\succ^N}$, $f_{\vec{\alpha}}$ and flipped veto do not agree.

We claim that for any two preference profiles $\succ_1^N$ and $\succ_2^N$ on which $f_{\vec{\alpha}}$ and flipped veto agree, it holds that $B_{\succ_1^N} \cap B_{\succ_2^N} = \emptyset$. Indeed, assume that there exists $\succ^N \in B_{\succ_1^N} \cap B_{\succ_2^N}$. Assume first that the winner in both profiles is x^*. It cannot be the case that the same alternative was switched with x^* in order to obtain $\succ^N$ from both $\succ_1^N$ and $\succ_2^N$—that would imply $\succ_1^N$ and $\succ_2^N$ are identical. Therefore, assume w.l.o.g. that x_1 was switched with x^* in $\succ_1^N$ (only in the rankings of voters that did not rank x^* last), and x_2 was switched with x^* in $\succ_2^N$. But this means that both x_1 and x_2 are winners in $\succ^N$ under $f_{\vec{\alpha}}$ (by the fact that x^* was a winner in both $\succ_1^N$ and $\succ_2^N$)—a contradiction.

In addition, in any two preference profiles $\succ_1^N$ and $\succ_2^N$ such that

$$f_{\vec{\alpha}}(\succ_1^N) = \text{flipped veto}(\succ_1^N) = x^*,$$

and

$$f_{\vec{\alpha}}(\succ_2^N) = \text{flipped veto}(\succ_2^N) = x^{**},$$

it holds that $B_{\succ_1^N} \cap B_{\succ_2^N} = \emptyset$, as flipped veto elects x^* in all profiles in $B_{\succ_1^N}$, but elects x^{**} in all profiles in $B_{\succ_2^N}$.

It follows that for every preference profile on which $f_{\vec{\alpha}}$ and flipped veto agree, there are at least $m-1$ distinct profiles on which the two voting rules disagree; this proves the proposition. □

We shall now formulate our main result for this Section. The theorem states that almost every voting rule cannot be approximated by a factor better than $\frac{1}{2}$ by any small family of voting rules. We shall subsequently see that the theorem holds for small voting trees as well as scoring rules.

[7] It cannot be the case that all voters ranked x^* last, by our tie-breaking assumption with respect to $f_{\vec{\alpha}}$.

Theorem 5.3. *Let $\mathcal{R}_m^n$ be a family of voting rules of size exponential in n and m, and let $\epsilon, \delta > 0$. For large enough values of n and m, at least a $(1-\delta)$-fraction of the voting rules $f : \mathcal{L}^n \to \{x_1, \ldots, x_m\}$ satisfy the following property: no voting rule in $\mathcal{R}_m^n$ is a $(1/2+\epsilon)$-approximation of f.*

Proof. We will surround each voting rule $f \in \mathcal{R}_m^n$ with a "ball" $B(f)$, which contains all the voting rules for which f is a $(1/2+\epsilon)$-approximation. We will then show that the union of all these balls covers at most a δ-fraction of the set of the space of voting rules. This implies that for at least a $(1-\delta)$-fraction of the voting rules, no scoring rule is a $(1/2+\epsilon)$-approximation.

For a given f, what is the size of $B(f)$? As there are $(m!)^n$ possible preference profiles, the ball contains rules that do not agree with f on at most $(1/2-\epsilon)(m!)^n$ preference profiles. For a profile on which there is disagreement, there are m options to set the image under the disagreeing rule.[8] Therefore,

$$|B(f)| \leq \binom{(m!)^n}{(1/2-\epsilon)(m!)^n} m^{(1/2-\epsilon)(m!)^n}. \tag{12}$$

How large is this expression? Let $B'(f)$ be the set of all voting rules that disagree with f on *exactly* $(1/2+\epsilon)(m!)^n$ preference profiles. It holds that

$$\begin{aligned} |B'(f)| &= \binom{(m!)^n}{(1/2+\epsilon)(m!)^n} (m-1)^{(1/2+\epsilon)(m!)^n} \\ &= \binom{(m!)^n}{(1/2-\epsilon)(m!)^n} ((m-1)^{1+2\epsilon})^{1/2(m!)^n} \\ &\geq \binom{(m!)^n}{(1/2-\epsilon)(m!)^n} m^{1/2(m!)^n}, \end{aligned} \tag{13}$$

where the last inequality holds for a large enough m. But since the total number of voting rules, $m^{(m!)^n}$, is greater than the number of rules in $B'(f)$, we have:

$$\frac{m^{(m!)^n}}{B(f)} \geq \frac{B'(f)}{B(f)} \geq \frac{\binom{(m!)^n}{(1/2-\epsilon)(m!)^n} m^{1/2(m!)^n}}{\binom{(m!)^n}{(1/2-\epsilon)(m!)^n} m^{(1/2-\epsilon)(m!)^n}} = m^{\epsilon(m!)^n}. \tag{14}$$

Therefore

$$B(f) \leq \frac{m^{(m!)^n}}{m^{\epsilon(m!)^n}} = m^{(1-\epsilon)(m!)^n}. \tag{15}$$

If the union of balls is to cover at least a δ-fraction of the set of voting rules, we must have $|\mathcal{R}_m^n| \cdot m^{(1-\epsilon)(m!)^n} \geq \delta \cdot m^{(m!)^n}$; equivalently, it must hold that $|\mathcal{R}_m^n| \geq \delta \cdot m^{\epsilon(m!)^n}$. However, by the assumption $|\mathcal{R}_m^n|$ is only exponential in n and m (rather than double exponential), so for large enough values of n and m, the above condition does not hold. □

Notice that the number of distinct voting trees with at most k leaves, as voting rules $f : \mathcal{L}^N \to A$ where $|A| = m$, is bounded from above for any number of voters n by the expression given in Lemma 4.5, namely $k \cdot m^k \cdot C_{k-1}$. Therefore, we have as a corollary from Theorem 5.3:

Corollary 5.4. *For large enough values of n and m, almost all voting rules cannot be approximated by $\mathcal{V}_m^{(k)}$, k polynomial in m, to a factor better than $\frac{1}{2}$.*

[8] This reasoning also takes into account voting rules that agree with f on more than $(1/2+\epsilon)(m!)^n$ profiles.

In order to obtain a similar corollary regarding scoring rules, we require the following lemma, which may be of independent interest.

Lemma 5.5. *There exists a polynomial $p(n,m)$ such that for all $n,m \in \mathbb{N}$, $|\mathcal{S}_m^n| \leq 2^{p(n,m)}$.*

Proof. It is true that there are an infinite number of ways to choose the vector $\vec{\alpha}$ that defines a scoring rule. Nevertheless, what we are really interested in is the number of *distinct* scoring rules. For instance, if $\vec{\alpha}^1 = 2\vec{\alpha}^2$, then $f_{\vec{\alpha}^1} \equiv f_{\vec{\alpha}^2}$, i.e., the two vectors define the same voting rule.

It is clear that two scoring rules $f_{\vec{\alpha}^1}$ and $f_{\vec{\alpha}^2}$ are distinct only if the following condition holds: there exist two alternatives $x_{j_1}, x_{j_2} \in C$, and a preference profile $\succ^N$, such that $f_{\vec{\alpha}^1}(\succ^N) = x_{j_1}$ and $f_{\vec{\alpha}^2}(\succ^N) = x_{j_2}$. This holds only if there exist two alternatives x_{j_1} and x_{j_2} and a preference profile $\succ^N$ such that under $\vec{\alpha}^1$, x_{j_1}'s score is strictly greater than x_{j_2}'s, and under $\vec{\alpha}_2$, either x_{j_2}'s score is greater or the two alternatives are tied, and the tie is broken in favor of x_{j_2}.

Now, assume $\succ^N$ induces rankings $\vec{\pi}_{j_1}$ and $\vec{\pi}_{j_2}$. The conditions above can be written as

$$\sum_l \pi_{j_1,l}\alpha_l^1 > \sum_l \pi_{j_2,l}\alpha_l^1, \tag{16}$$

$$\sum_l \pi_{j_1,l}\alpha_l^2 \leq \sum_l \pi_{j_2,l}\alpha_l^2, \tag{17}$$

where the inequality is an equality only if ties are broken in favor of x_{j_2}, i.e., if $l_0 = \min\{l : \pi_{j_1,l} \neq \pi_{j_2,l}\}$, then $\pi_{j_1,l} < \pi_{j_2,l}$.[9]

Let $\vec{\pi}_\Delta = \vec{\pi}_{j_1} - \vec{\pi}_{j_2}$. As in the proof of Lemma 3.6, equations (16) and (17) can be concisely rewritten as

$$\vec{\pi}_\Delta \cdot \vec{\alpha}^1 > 0 \geq \vec{\pi}_\Delta \cdot \vec{\alpha}^2, \tag{18}$$

where the inequality is an equality only if the first nonzero position in $\vec{\pi}_\Delta$ is negative.

In order to continue, we opt to reinterpret the above discussion geometrically. Each point in $\mathbb{R}^m$ corresponds to a possible choice of parameters $\vec{\alpha}$. Now, each possible choice of $\vec{\pi}_\Delta$ is the normal to a hyperplane. These hyperplanes partition the space into cells: the vectors in the interior of each cell agree on the signs of dot products with all vectors $\vec{\pi}_\Delta$. More formally, if $\vec{\alpha}_1$ and $\vec{\alpha}_2$ are two points in the interior of a cell, then for any vector $\vec{\pi}_\Delta$, $\vec{\pi}_\Delta \cdot \vec{\alpha}^1 > 0 \Leftrightarrow \vec{\pi}_\Delta \cdot \vec{\alpha}^2 > 0$. By equation (18), this implies that any two scoring rules $f_{\vec{\alpha}^1}$ and $f_{\vec{\alpha}^2}$, where $\vec{\alpha}^1$ and $\vec{\alpha}^2$ are in the interior of the same cell, are identical.

What about points residing in the intersection of several cells? These vectors always agree with the vectors in one of the cells, as ties are broken according to rankings induced by the preference profile, i.e., according to the parameters that define our hyperplanes. Therefore, the points in the intersection can be conceptually annexed to one of the cells.

So, we have reached the conclusion that the number of distinct scoring rules is at most the number of cells. Hence, it is enough to bound the number of cells; we claim this number is exponential in n and m. Indeed, each $\vec{\pi}_\Delta$ is an m-vector, in which every coordinate is an integer in the set $\{-n, -n+1, \ldots, n-1, n\}$. It follows that there are at most $(2n+1)^m$ possible hyperplanes. It is known [7] that given k hyperplanes in d-dimensional space, the number of cells is at most $O(k^d)$. In our case, $k \leq (2n+1)^m$ and $d = m$, so we have obtained a bound of:

$$((2n+1)^m)^m \leq (3n)^{m^2} = \left(2^{\log 3n}\right)^{m^2} = 2^{m^2 \log 3n}. \tag{19}$$

[9] W.l.o.g. we disregard the case where $\vec{\pi}_{j_1} = \vec{\pi}_{j_2}$; the reader can verify that taking this case into account multiplies the final result by an exponential factor at most.

□

Remark 5.6. This lemma implies, according to Lemma 2.4, that there exists a polynomial $p(n, m)$ such that for all $n, m \in \mathbb{N}$, $D_G(\mathcal{S}_m^n) \leq p(n, m)$. However, we have already obtained a tighter upper bound of m.

Finally, using Theorem 5.3 and Lemma 5.5 we obtain:

Corollary 5.7. *For large enough values of n and m, almost all voting rules cannot be approximated by $\mathcal{S}_m^n$ to a factor better than $\frac{1}{2}$.*

Remark 5.8. Proposition 5.2 can seemingly be circumvented by removing the requirement that in a scoring rule defined by a vector $\vec{\alpha}$, $\alpha_l \geq \alpha_{l+1}$ for all l. Indeed, flipped veto is essentially a scoring rule with $\alpha_m = 1$ and $\alpha_l = 0$ for all $l \neq m$. However, the constant voting rule which always elects the same alternative has the same inapproximability ratio, even when this property of scoring rules is not taken into account. Moreover, Corollary 5.7 also holds when scoring rules are not assumed to satisfy this property.

6 Discussion

We have demonstrated the possibility of learning scoring rules and small voting trees. We have argued that, given a black box specification of the choice criteria of the society, learning from examples allows one to efficiently (albeit approximately) design such rules. The black box reflects some ideal voting rule the designer has in mind, which satisfies, for instance, different desirable properties. The designer thus essentially translates a cumbersome representation of a voting rule (hidden within the black box) to a concisely represented voting rule which is easy to understand and apply.

In Section 5 we have explored the possibility of extending our approach to the setting where the designer has in mind some general voting rule, rather than a scoring rule or a voting tree, and we would like to find a scoring rule or voting tree that is as close as possible. Technically, our learning-theoretic results basically hold (up to polynomial factors in the sample complexity) in this setting, although the situation is more difficult in terms of computational complexity.

Unfortunately, it turns out (Corollaries 5.4 and 5.7) that many voting rules cannot be approximated, neither by using scoring rules nor by small voting trees. However, this negative result relied implicitly on assuming a uniform distribution over profiles. More importantly, it might be the case that some of the important families of voting rules can be approximated by scoring rules or small voting trees. Therefore, we do not rule out at this point the application of our approach to designing general voting rules by directly learning scoring rules or small voting trees that approximate them.

Criticisms of our approach. A possible concern, given Corollaries 5.4 and 5.7, is with our general motivation. Indeed, if we assume that the designer has in mind, say, a scoring rule, it can be argued that the designer must be aware of this fact, and must have knowledge of the parameters of the rule. However, recall that the class of scoring rules is exactly the class of anonymous, neutral, and consistent voting rules [28]. Hence, if the designer selects winners in any way that satisfies these three desiderata, a scoring rule with unknown parameters would be obtained.

A similar case can be made for voting trees. The underlying assumption behind the literature on implementation by voting trees (see, e.g., [10] and the references therein) is that voting trees are an abstract model of decision making, and that many voting rules can in fact be represented as voting trees, even if this transformation is not obvious. For example, the Copeland rule, that selects an alternative that beats the largest number of alternatives in pairwise elections, can be represented as an elaborate voting tree if there are up to seven alternatives [25]. Hence, the designer might be using a voting rule that can be represented as a voting tree, but might be unaware of the exact representation.

Let us discuss two additional possible criticisms regarding our general setting. First, notice that in multiagent environments, the number of alternatives m can be large; for example, if the agents are voting on joint plans [9], then the number of alternatives might be significantly larger than the number of agents. Hence, complexity results that depend on the number of alternatives are meaningful.

Second, it has been suggested that the designer might find it easier to express the ethical properties that are considered mandatory, rather than express the voting rule by examples. We argue that this is rarely the case. Indeed, it is very difficult to concisely represent properties in computational settings; a universal, agreed-upon language would have to be used, and it is hard to imagine how one would go about creating such a language. On the other hand, specifying a voting rule by (a polynomial number of) examples provides a concise description of the voting rule, and, as we have shown, can lead to a close approximation.

Future work. We mention two directions for future research. First, imagine the following scenario: the designer has in mind a huge voting tree, and would like to know whether there exists a smaller voting tree that implements the same voting rule. The same goes for scoring rules, e.g., the designer might have in mind a scoring rule with huge values for components of the vector $\vec{\alpha}$. This is a setting closely related to ours, but our results do not hold in the alternative setting.

Second, it might prove interesting to study the learnability of larger families of voting rules that have a concise representation. One compelling example is the class of *generalized scoring rules* recently proposed by Xia and Conitzer [27].

7 Acknowledgments

The authors wish to thank anonymous AIJ, AAAI, and AAMAS reviewers, for their very helpful comments. This work was partially supported by Israel Science Foundation grant #898/05.

References

[1] J. Bartholdi, C. A. Tovey, and M. A. Trick. How hard is it to control an election. *Mathematical and Computer Modelling*, 16:27–40, 1992.

[2] E. Beigman and R. Vohra. Learning from revealed preference. In *Proceedings of the 7th ACM Conference on Electronic Commerce (ACM-EC)*, pages 36–42, 2006.

[3] V. Conitzer and T. Sandholm. Complexity of mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 103–110, 2002.

[4] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 781–788, 2003.

[5] V. Conitzer and T. Sandholm. An algorithm for automatically designing deterministic mechanisms without payments. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 128–135, 2004.

[6] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54:1–33, 2007.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.

[8] E. Elkind and H. Lipmaa. Small coalitions cannot manipulate voting. In *Proceedings of the Annual Conference on Financial Cryptography (FC)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[9] E. Ephrati and J. S. Rosenschein. A heuristic technique for multiagent planning. *Annals of Mathematics and Artificial Intelligence*, 20:13–67, 1997.

[10] F. Fischer, A. D. Procaccia, and A. Samorodnitsky. A new perspective on implementation by voting trees. Manuscript, 2008.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[12] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: the anatomy of a recommender system. In *Proceedings of the 3rd Annual Conference on Autonomous Agents (AGENTS)*, pages 434–435, 1999.

[13] T. Haynes, S. Sen, N. Arora, and R. Nadella. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the 1st Annual Conference on Autonomous Agents (AGENTS)*, pages 308–315, 1997.

[14] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. *Journal of Computer and System Sciences*, 73(1):73–83, 2007.

[15] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.

[16] G. Kalai. Learnability and rationality of choice. *Journal of Economic Theory*, 113(1):104–117, 2003.

[17] S. Lahaie and D. C. Parkes. Applying learning algorithms to preference elicitation. In *Proceedings of the 5th ACM Conference on Electronic Commerce (ACM-EC)*, pages 180–188, 2004.

[18] J. Lang, M.-S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Winner determination in sequential majority voting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1372–1377, 2007.

[19] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[20] B. K. Natarajan. *Machine Learning: A Theoretical Approach*. Morgan Kaufmann, 1991.

[21] A. D. Procaccia and J. S. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research*, 28:157–181, February 2007.

[22] A. D. Procaccia, J. S. Rosenschein, and A. Zohar. Multi-winner elections: Complexity of manipulation, control and winner-determination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1476–1481, 2007.

[23] A. D. Procaccia, A. Zohar, Y. Peleg, and J. S. Rosenschein. Learning voting trees. In *Proceedings of the 22nd AAAI Conference on AI (AAAI)*, pages 110–115, 2007.

[24] A. D. Procaccia, A. Zohar, and J. S. Rosenschein. Automated design of scoring rules by learning from examples. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 951–958, 2008.

[25] S. Srivastava and M. A. Trick. Sophisticated voting rules: The case of two tournaments. *Social Choice and Welfare*, 13:275–289, 1996.

[26] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2nd edition, 2001.

[27] L. Xia and V. Conitzer. Generalized Scoring Rules and the frequency of coalitional manipulability. In *Proceedings of the 9th ACM Conference on Electronic Commerce (ACM-EC)*, pages 109–118, 2008.

[28] H. P. Young. Social choice scoring functions. *SIAM Journal of Applied Mathematics*, 28(4), 1975.

[29] M. Zuckerman, A. D. Procaccia, and J. S. Rosenschein. Algorithms for the coalitional manipulation problem. In *The 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 277–286, 2008.