Preferences Single-Peaked on a Tree: Sampling and Tree Recognition

Jakub Sliwinski[1] and **Edith Elkind**[2]
[1]ETH Zurich
[2]University of Oxford
jsliwinski@ethz.ch, elkind@cs.ox.ac.uk

Abstract

In voting theory, impossibility results and computational hardness results are often circumvented by recognising that voters' preferences are not arbitrary, but lie within a restricted domain. Uncovering the structure of the underlying domain often provides useful insights about the nature of the alternative space, and may be helpful in identifying a collective choice. Preferences single-peaked on a tree are an example of a relatively broad domain that nonetheless exhibits several desirable properties. We consider the setting where agents' preferences are independently sampled from rankings that are single-peaked on a given tree, and study the problem of reliably identifying the tree that generated the observed votes. We test our algorithm empirically; to this end, we develop a procedure to uniformly sample preferences that are single-peaked on a given tree.

1 Introduction

In multi-agent systems agents often have to make group decisions by reporting their preferences over the available alternatives (candidates): each agent specifies her ranking of the alternatives, and the decision is determined by an agreed-upon preference aggregation rule. In general, preference aggregation is a complex problem: if each voter may submit any ranking of the alternatives, then no voting rule can simultaneously enjoy a small number of simple and highly desirable properties [Arrow, 1951; Gibbard, 1973; Satterthwaite, 1975]. To make matters worse, many voting rules are not computationally tractable [Bartholdi III *et al.*, 1989; Brandt *et al.*, 2016].

However, agents' preferences are hardly arbitrary, and in many circumstances the votes are intrinsically restricted to a certain domain. Hence, we may be interested in recognising that domain to gain insight about the nature of the alternative space. Many domain restrictions, when uncovered, can be used to circumvent the problems we mentioned. Some examples of domain restrictions that have proved to be useful in this context are single-crossing preferences [Mirrlees, 1971; Roberts, 1977] and value-restricted preferences [Sen, 1966; Sen and Pattanaik, 1969], which include single-peaked [Black, 1948] and group-separable preferences [Inada, 1964; 1969]. For example, a collection of votes (a *profile*) is single-peaked with respect to a certain order (an axis) of candidates, if for every vote v in this collection and every candidate c, all candidates that lie between c and the top choice of v on the axis are ranked higher than c by v. If preferences are assumed to be single-peaked, Arrow's impossibility theorem does not apply, there are strategy-proof voting rules [Moulin, 1980; 1991], and many problems that are NP-hard in general become tractable [Brandt *et al.*, 2015; Walsh, 2007; Faliszewski *et al.*, 2011; Cornaz *et al.*, 2012].

If the candidates are arranged on a tree rather than a line, we obtain the domain of preferences single-peaked on a tree [Demange, 1982]; more precisely, a profile is single-peaked on a tree if it is single-peaked when restricted to any path in the tree. This domain subsumes the domain of single-peaked preferences and retains some of its desirable properties: for example, profiles that are single-peaked on a tree always have weak Condorcet winners [Demange, 1982], and, for certain trees, many hard voting rules become tractable [Yu *et al.*, 2013; Peters and Elkind, 2016] and preference elicitation becomes easier [Dey and Misra, 2016]. An example of a setting with this type of preferences is facility location on acyclic road network; for further examples, see [Demange, 1982].

A specific scenario where it may be useful to know whether the alternative space has a tree-like structure is multiwinner voting [Faliszewski *et al.*, 2017]. Suppose that voters have to select a fixed-size subset of alternatives so that each voter likes at least one alternative in this set. It has been argued that the best multiwinner voting rule for this goal is the Chamberlin–Courant rule [Chamberlin and Courant, 1983]. While it is NP-hard to compute a winning set under this rule, this problem becomes polynomial-time solvable when voters' preferences are single-peaked on a 'nice' tree [Peters and Elkind, 2016]. Consider now this situation from the perspective of the voting authorities: they would like to use the Chamberlin–Courant rule as long as it is computationally feasible, since this rule is axiomatically justified, and otherwise they would prefer to use an efficient procedure that outputs approximately optimal committees [Skowron *et al.*, 2015], or at least reserve more time for the aggregation procedure. However, the authorities need to commit to a voting procedure before the general election, so they need to decide whether voters' preferences are likely to be single-peaked on a nice tree without eliciting all votes.

One way to achieve that is to draw a small random sample of voters and check if this sample is single-peaked on a nice tree. Indeed, there are polynomial-time algorithms for checking whether a given collection of votes is single-peaked on a tree and identifying some such tree [Trick, 1989]; in fact, it is possible to compactly represent all suitable trees, and pick a tree with desirable properties [Peters and Elkind, 2016]. However, in general, such a tree is not unique. This presents a problem in our setting: if the election authorities obtain a sample that is consistent with several different trees, with some, but not all of these trees having the required structure, they cannot reasonably expect that, once all votes are elicited, the resulting instance would be single-peaked on a nice tree. One can continue to sample votes until there is only one tree that is consistent with the votes, but, as we will show, this may require a number of samples that is exponential in the number of alternatives. Thus, it is desirable to have a procedure that identifies a tree that is most likely to have generated a given sample of votes, with high probability.

Our contribution. In this paper, we consider the setting where votes are sampled independently at random among all rankings that are single-peaked on a given tree. We are interested in estimating the number of votes that is sufficient to uniquely identify the underlying tree. We argue that we may need exponentially many samples to identify the tree with certainty if the votes are drawn from the uniform distribution over preferences that are single-peaked on that tree. However, we obtain a PAC-learning style result showing that we only need a small number of samples to correctly guess the tree with high probability. We complement our theoretical results with empirical analysis. To conduct our experiments, and specifically to provide input for our learning algorithms, we design an efficient algorithm for uniformly sampling preferences that are single-peaked on a given tree; Walsh [2015] has proposed such an algorithm for preferences that are single-peaked on a line [see also Conitzer, 2009], but extending this result to trees is a significant challenge.

2 Preliminaries

A *vote* over a finite set of candidates $C = \{1, \dots, m\}$ is a strict total order over C. Given a vote v and two candidates $x, y \in C$, we write $x >_v y$ to indicate that v ranks x above y. We denote the candidate ranked in position k in v by $v[k]$, and let $v_{[1:k]} = \{v[1], \dots, v[k]\}$; also, we sometimes write $v[-1]$ to denote the lowest-ranked candidate in v. A *profile* over C is a list $V = (v_1, \dots, v_n)$ of votes over C. For each $c \in C$, the *Borda score* $B_v(c)$ of c in a vote v is the number of candidates ranked lower than c in v. The Borda score of c in a profile V is $B_V(c) = \sum_{v \in V} B_v(c)$. For a $C' \subseteq C$, we denote by $v \upharpoonright_{C'}$ the vote obtained by restricting v to C'.

A profile V is *single-peaked (on a line)* if there is a linear order $\lhd$ on C such that, for all $v \in V$ and every pair of candidates $x, y \in C$ such that $v[1] \lhd x \lhd y$ or $y \lhd x \lhd v[1]$ we have $x >_v y$. The profile V is *single-peaked on T*, where T is a tree with vertex set C, if V is single-peaked when restricted to the vertex set of every path of T. Equivalently, V is single-peaked on T if for every $v \in V$ and each $k = 1, \dots, |C|$, the set $v[1:k]$ induces a subtree of T. Given a profile V, we denote the set of all trees T such that V is single-peaked on T by $\mathcal{T}(V)$. We say that trees in $\mathcal{T}(V)$ are *suitable* for V.

3 Sampling Algorithm

Given a tree T, let $\mathcal{U}(T)$ be the uniform distribution on the set of votes $\{v : v \text{ is single peaked on } T\}$. In this section we describe a procedure for sampling a vote from $\mathcal{U}(T)$. This algorithm will be used in our experiments (Section 5) to empirically evaluate the learning algorithms described in Section 4.

We split the algorithm into two subroutines:

- $P_1(T)$, determining $v[1]$;
- $P_2(T, c)$, determining $v[2], \dots, v[m]$ given $v[1] = c$.

We present these subroutines in reverse order.

Sampling a vote with given $\boldsymbol{v[1]}$ Fix a tree $T = (C, E)$, $C = \{1, \dots, m\}$, and a candidate $c \in C$. Let $S_2(T, c) = \{v$ is single-peaked on $T : v[1] = c\}$. We describe a randomised procedure $P_2(T, c)$ that samples elements of $S_2(T, c)$ uniformly at random. From now on we view T as rooted at c; in particular, we say that a vertex b is a *child* of a vertex a if a is the first vertex on the path from b to c. The following simple observation will be useful for our analysis.

Lemma 1. *A vote v with $v[1] = c$ is single-peaked on T if and only if it ranks every candidate $a \in C$ above its children.*

We say that a subtree of T is a *vine descending from y* if this subtree consists of a child x of y and all descendants of x, and only contains one leaf. Suppose first that T consists of the root c and two vines that descend from c. Denote these vines by p_1 and p_2, and suppose that $|p_1| = m_1$, $|p_2| = m_2$, where $|p|$ denotes the number of nodes in a vine p. Note that $m = m_1 + m_2 + 1$. There is a one-to-one correspondence between the sets $S_2(T, c)$ and $\mathcal{A} = \{A \subset \{2, 3, \dots, m\} : |A| = m_1\}$: we can view a set $A \in \mathcal{A}$ as the set of positions in the vote where candidates in p_1 can be placed, so that the candidates in p_2 are placed in the remaining m_2 positions (note that the order of these candidates is uniquely determined by Lemma 1). We have $|S_2(T, c)| = |\mathcal{A}| = \binom{m-1}{m_1}$. We will refer to the following procedure as *merging the vines p_1 and p_2 in T*: given p_1 and p_2, pick an element $A \in \mathcal{A}$ uniformly at random, let v be the vote in $S_2(T, c)$ that corresponds to A, and replace T with the path $(C, \{(c, v[2]), (v[2], v[3]), \dots, (v[m-1], v[m])\})$. We denote this procedure by $M(c, p_1, p_2)$.

We are now ready to consider the general case, where T is a tree rooted at c.

Lemma 2. *Algorithm 1 performs $P_2(T, c)$.*

Algorithm 1 Sample v single-peaked on T with $v[1] = c$

```
1: while $T$ is not a path starting at $c$ do
2:     Find a vertex $a$ with at least two vines $p_1, p_2$ descending from $a$.
3:     Perform $M(a, p_1, p_2)$.
4: end while
5: return The unique vote in $S_2(T, c)$.
```

Proof. The **while** loop terminates after $m-1$ iterations, since each iteration decreases the number of leaves in T by one, and when there is only one leaf, T is a path.

Further, any vote returned by Algorithm 1 is single-peaked on T, since merging the vines in line 3 preserves descendancy: if a is a descendant of b before the merge, this is also the case after the merge.

Conversely, every vote v that is single-peaked on T can be obtained by a sequence of merges; the order of merges can be arbitrary, but during every merge there is a unique set of indices $A \in \mathcal{A}$ that corresponds to v. Each element of $\mathcal{A}$ is equally likely, and so is every vote single-peaked on T. □

Algorithm 1 runs in polynomial time, but is too slow to be used in practice. However, we can use it as a reasoning tool to arrive at a more efficient solution. In the remainder of this section, we explore a greedy approach that outputs the candidates in v one by one. The subroutine *highestAncestor*(d, R) in this algorithm returns the highest ancestor of d that is an element of R.

Before we present a proof of correctness of Algorithm 2, we briefly describe this algorithm in a less formal way. The i-th iteration of the **while** loop can be seen as follows.

- There is a pool of disjoint subtrees of T; $v[i]$ is selected among the roots of these subtrees.
- The probability that the root of a subtree is selected as $v[i]$ is proportional to the size of the subtree.
- When a candidate a is picked as $v[i]$, the a's subtree is replaced in the pool with the subtrees of a's children.

Lemma 3. *Algorithm 2 performs $P_2(T, c)$.*

Proof. Consider some vertex u that is a child of c. Let m_1 be the size of the subtree rooted at u; we would like to compute the probability p that Algorithm 1 returns $v[2] = u$. Consider an execution of Algorithm 1 where we merge vines in such an order that at the last step we merge p_1 and p_2 and p_1 begins with u; then p only depends on what happens during the last merge. Let $m_1 = |p_1|$. The number of votes with $v[2] = u$ is then given by $\binom{m-2}{m_1-1}$, and the total number of possible votes is $\binom{m-1}{m_1}$. Thus, the probability of u being second is

$$\frac{\binom{m-2}{m_1-1}}{\binom{m-1}{m_1}} = \frac{m_1}{m-1}.$$

Since m_1 is the size of the subtree of T rooted at u, the probability that a random vertex from $C \setminus \{c\}$ lies in u's subtree equals the probability that $v[2]$ equals u.

Once $v[2]$ is determined, $v[3]$ may be any child of c except $v[2]$, or any child of $v[2]$. The same reasoning can be repeated with respect to $v[i]$, for $i = 3, 4, \dots, m$, to the conclusion that for each i, the probability that $v[i] = z$ is proportional to the size of the subtree rooted at z, for any z in the set

$$\bigcup_{k=1}^{i-1} \text{children}(v[k]) \setminus \{v[1], ..., v[i-1]\}.$$

Hence, Algorithm 2 determines $v[i]$ at every iteration of the loop, preserving the distribution of results of Algorithm 1. □

Algorithm 2 Sample v single-peaked on T with $v[1] = c$

```
1: $v[1] \leftarrow c$
2: $R \leftarrow C \setminus \{c\}$
3: $i \leftarrow 2$
4: while $R \neq \emptyset$ do
5:     $d \leftarrow$ uniformly random element of $R$
6:     $a \leftarrow$ highestAncestor$(d, R)$
7:     $R \leftarrow R \setminus \{a\}$
8:     $v[i] \leftarrow a$
9:     $i \leftarrow i + 1$
10: end while
11: return $v$
```

In the interest of space we do not discuss how to implement the subroutines in Algorithm 2 efficiently; the implementation in our experiments has time complexity $O(m \log m)$.

Determining $\boldsymbol{v}[1]$ To use Algorithm 2, we need to determine $v[1]$. The following lemma will be crucial for our analysis.

Lemma 4. *Consider an edge xy of a tree T. Let T_x and T_y be the connected components of the graph $T \setminus \{xy\}$, with $x \in T_x$, $y \in T_y$. Let v be a vote sampled from $\mathcal{U}(T)$. Then*

$$\frac{\Pr(v[1] = x)}{\Pr(v[1] = y)} = \frac{|T_x|}{|T_y|}.$$

Proof. We have

$$\frac{\Pr(v[1] = x)}{\Pr(v[1] = y)} = \frac{|\{v \text{ is single-peaked on } T : v[1] = x\}|}{|\{v \text{ is single-peaked on } T : v[1] = y\}|}.$$

Consider two executions of Algorithm 1, E_x and E_y, with, respectively, x and y being selected as $v[1]$. Note that for E_x the root is x and one of its children is y, and for E_y the opposite is true. Recall that the order in which vines are merged is irrelevant for the distribution of results. In both cases, let the order of merges be such that all merges happen inside of T_x or inside of T_y, until only two vines descending from the root remain. Up to this point, the two executions are the same, and the number z of possible results until now is the same. The last step of E_x and E_y is merging the vines of lengths $|T_x| - 1, |T_y|$, and $|T_x|, |T_y| - 1$ respectively. The number of possible results in the two cases is $z\binom{|T_x|+|T_y|-1}{|T_x|-1}$ and $z\binom{|T_x|+|T_y|-1}{|T_y|-1}$, respectively, hence

$$\frac{\Pr(v[1] = x)}{\Pr(v[1] = y)} = \frac{z\binom{|T_x|+|T_y|-1}{|T_x|-1}}{z\binom{|T_x|+|T_y|-1}{|T_y|-1}} = \frac{|T_x|}{|T_y|}.$$

□

Lemma 5. *Algorithm 3 computes* $\Pr(v[1] = i)$ *for each candidate $i \in C$.*

Proof. By Lemma 4, the values $P(i)$ computed in Line 5 are proportional to $\Pr(v[1] = i)$, and Line 6 normalises them. □

Algorithm 3 Compute $\Pr(v[1] = k)$ for $k = 1, \dots, m$ given the tree $T = (\{1, \dots, m\}, E)$

```
1: Root $T$ at 1.
2: Renumber the candidates so that for each $i \neq 1$ it holds
   that parent$(i) < i$.
3: For each candidate $i$, compute the size $s_i$ of the subtree
   rooted at $i$.
4: $P(1) = 1$                                  ▷ reference value.
5: For $i = 2, \dots, m$ let $P(i) \leftarrow \frac{s_i \cdot P(\text{parent}(i))}{m - s_i}$
6: For $i = 1, \dots, m$ let $\Pr(v[1] = i) = \frac{P(i)}{\sum_{k=1}^{m} P(k)}$
7: return
```

Algorithm 3 operates on values that may be exponential in m. Hence, if we assume that an arithmetic operation on numbers a and b takes time $O(\max\{\log a, \log b\})$, the running time of Algorithm 3 can be bounded as $\Theta(m^2)$.

Sampling votes We note that, given a tree T, Algorithm 3 needs to be run just once, no matter how many votes we have to sample. Then, $v[1]$ can be chosen based on the probabilities computed, and Algorithm 3 can be run repeatedly, sampling each vote in time $O(m \log m)$. This procedure will be referred to as the Sampling Algorithm.

Combining Lemma 3 and Corollary 5 gives the following result.

Theorem 6. *Given a tree T, we can efficiently sample votes from $\mathcal{U}(T)$.*

4 Tree Identification

In this section we investigate the problem of identifying the tree that describes the vote domain based on the observed votes. The central question is: how many votes do we need to examine in order to guess the tree? We consider two variants of this question, which depend on whether we need to determine the tree with certainty or with high probability.

Attachment digraph A useful tool in our analysis is the *attachment digraph*, defined by Peters and Elkind [2016] building on the ideas of Trick [1989] and Yu *et al.* [2013]. Given a profile V over a candidate set C, its *attachment digraph* is a digraph $D = (C, A)$ that is built by Algorithm 4; it relies on the subroutine $B(v, c)$, which takes as input a vote v and candidate c, and returns the set of candidates that c can be attached to as a leaf in trees suitable for v.

$$B(v, c) = \begin{cases} \{c' : c' >_v c\} & \text{if } v[1] \neq c \\ \{v[2]\} & \text{if } v[1] = c \end{cases}$$

The attachment digraph offers a compact description of the set $\mathcal{T}(V)$, in the following sense. It can be shown that D has at most two sinks. To obtain the *pointed attachment digraph* D' we set $D' = D$, and then, if D has two sinks, we add an arc between them, directed arbitrarily. By definition, D' has at most one sink s. We say that a vertex $z \in C$ is *free* if $|\{x : zx \in A\}| > 1$ and *forced* otherwise. Peters and Elkind [2016] prove that any tree in $\mathcal{D}(V)$ can be obtained from D' by removing arcs so that the remaining graph contains exactly one arc of the form zx for each $z \in C \setminus \{s\}$, and forgetting the orientation.

Algorithm 4 Build attachment digraph $D = (C, A)$ of V

```
1: $D \leftarrow (C, \emptyset)$, the empty digraph on $C$
2: $C' \leftarrow C$
3: while $|C'| \geq 3$ do
4:     $L \leftarrow \{(v \upharpoonright_{C'})[-1] : v \in V\}$
5:     for each candidate $c \in L$ do
6:         $B_c = \bigcap_{v \in V} B(v \upharpoonright_{C'}, c)$
7:         if $B_c = \emptyset$ then
8:             return fail: $V$ not single-peaked on any tree
9:         else
10:            add arcs $cc'$ for each $c' \in B_c$ to $A$
11:        end if
12:    end for
13:    $C' \leftarrow C' \setminus L$
14: end while
15: return $D$
```

4.1 Tree Identification with Certainty

In general, to identify the tree T with certainty, we may need to sample exponentially many votes from $\mathcal{U}(T)$.

Theorem 7. *Suppose that a profile V is obtained by sampling n votes from $\mathcal{U}(T)$, where T is a path with m vertices. If $n < 2^{m-3}$ then $\Pr(|\mathcal{T}(V)| = 1) \leq \frac{1}{2}$.*

Proof. Consider a path T of the form 1–…–m. Note that $|\mathcal{T}(V)| = 1$ if and only if every vertex in the attachment digraph of T is forced. By the definition of $B(v, c)$ candidate 1 is free if candidates 2 and 3 are ranked higher than 1 in every vote. There are only two votes single-peaked on T in which 2 or 3 is ranked below 1, namely, $u = 1 > 2 > \cdots > m$ and $v = 2 > 1 > 3 > \cdots > m$. Hence, for T to be identified with certainly, the sampled profile needs to contain u or v. There are 2^{m-1} votes single-peaked on T (see, e.g., [Walsh, 2015]), so the probability of sampling u or v is $\frac{1}{2^{m-2}}$, and the probability that a profile of $n < 2^{m-3}$ votes contains neither u nor v is $\left(1 - \frac{1}{2^{m-2}}\right)^n > \frac{1}{2}$. □

The number of samples appears to be related to the diameter of the graph: e.g., if T is a star, we only need logarithmically many samples.

Theorem 8. *Suppose that a profile V is obtained by sampling n votes from $\mathcal{U}(T)$, where T is a star with m vertices. For every $\epsilon > 0$, if $n \geq 2 \log_2 m + \log_2 \frac{1}{\epsilon}$, then $\Pr(|\mathcal{T}(V)| = 1) \geq 1 - \epsilon$.*

4.2 Tree Identification with High Probability

Suppose that we have obtained a profile V by sampling n votes from $\mathcal{U}(T)$. If V is single-peaked with respect to several different trees, we cannot identify T with certainty; however, it may still be possible to identify T with high probability.

Example 9. Consider two trees on candidate set $\{1, \dots, m\}$: the first tree T_1 is a path of the form 1–2–…–m, and the second tree T_2 consists of the path 3–…–m, with both candidates 1 and 2 attached to 3 as leaves. Consider a sample of votes from $\mathcal{U}(T_1)$. If it does not contain votes u and v defined in the proof of Theorem 7, we cannot be sure that this sample was not drawn from $\mathcal{U}(T_2)$. However, if the votes are indeed sampled from T_1, then no voter ranks 2 last whereas in a sample of votes from $\mathcal{U}(T_2)$ candidates 1 and 2 are equally likely to be ranked last. Thus, if the sample is large, and 2 has never appeared in the last position, we can be reasonably confident that the underlying tree is T_1 rather than T_2.

Inspired by Example 9, we consider the problem of identifying T with high probability.

Given $\delta > 0$ and a profile of votes V sampled from $\mathcal{U}(T)$, our goal is to guess T correctly with probability at least $1-\delta$. We ask what is the smallest number of votes n that ensures this, as a function of $\frac{1}{\delta}$ and $|T|$.

Intuition. Consider the attachment digraph $D = (C, A)$ of V. Given a vertex c in T, let $PP(c) = \{u : cu \in A\}$ be the set of *potential parents* of c. Note that at most one vertex in $PP(c)$ is adjacent to c in T: while constructing the attachment digraph, we add arcs from c to vertices in $PP(c)$ only when c is ranked last in some vote, i.e., when c is a leaf in all trees consistent with the profile. We call this vertex the *real parent* of c; other vertices in $PP(c)$ are called the *false parents* of c. If c is forced, it has only one potential parent, which is certain to be its real parent. Now, let c be a free vertex in D. Our algorithm needs to guess which vertex in $PP(c)$ to specify as the real parent.

If u is a potential parent of c then by construction of the attachment digraph we have $u \in B(v, c)$ for all $v \in V$. Since c is free, we have $|B(v, c)| \geq 1$ for all $v \in V$, and hence by definition of $B(v, c)$ it holds that u is ranked above c in all votes in V. Suppose that c has two potential parents: a false parent f and the real parent p. It follows from the description of the Sampling Algorithm that c appears immediately after p with probability at least $\frac{1}{m}$. Thus, if vertices f, p, c are consistently ranked as $f > p > c$ in the votes, we can guess than p is the real parent of c.

Analysis of the algorithm. Consider Algorithm 5 where α is left to be determined. The operator *minBorda*(F, V) returns an element of $\{f : cf \in F\}$ that has the lowest Borda score in the profile V, breaking ties arbitrarily, and *fo*(C, A') is used to forget orientation, i.e., to turn a directed graph into an undirected graph.

Theorem 10. *For any tree T of size m and $\delta > 0$, if Algorithm 5 is given a profile V of $2m\alpha$ votes that are sampled from $\mathcal{U}(T)$, where $\alpha \geq 2\log m + \log\frac{1}{\delta}$, then the probability that the tree T' returned by the algorithm is equal to T is at least $1-\delta$.*

Algorithm 5 Guess T, given a profile V with $2m\alpha$ votes

```
1: $(C, A) \leftarrow$ PointedAttachmentDigraph$(V)$
2: $A' \leftarrow \emptyset$
3: for each candidate $c \in C$ do
4:     $F \leftarrow \{cf : cf \in A\}$    ▷ arcs to potential parents of $c$
5:     $A' \leftarrow A' \cup \{c, minBorda(F, V)\}$
6: end for
7: return $T' = fo(C, A')$
```

Algorithm 6 Try to sample a V such that $\neg E_{1,V} \wedge \neg E_{2,V}$ holds for given c, p and f

```
1: $V \leftarrow \emptyset$
2: for $k = 1, \ldots, 2m\alpha$ do
3:     $v$ is sampled from $\mathcal{U}(T)$
4:     $i \leftarrow 1$
5:     while $v[i] \notin \{p, f, c\}$ do
6:         $i \leftarrow i + 1$
7:     end while
8:     if $v[i] = c$ then
9:         return $\emptyset$
10:    end if
11:    if $v[i] = p$ then
12:        while $v[i] \neq f$ do
13:            if $v[i] = c$ then
14:                return $\emptyset$
15:            end if
16:            $i \leftarrow i + 1$
17:        end while
18:    end if
19:    $V \leftarrow V \cup \{v\}$
20: end for
21: if $B_V(f) > B_V(p)$ then
22:     return $\emptyset$
23: end if
24: return $V$
```

Proof sketch. Consider a free candidate c with parent p, and let f be a candidate in $C \setminus \{p\}$. We say that the algorithm *succeeds with* (c, f) if the arc cf does not appear in D of if f is a false parent of c in D, but cf is not chosen in line 5 of the algorithm. Clearly, if for each free candidate c and each candidate f that is not c's real parent the algorithm succeeds with (c, f), then the returned tree is equal to T.

We will argue that for each such pair (c, f) the algorithm succeeds with probability at least $1 - e^{-\alpha}$; since there are at most m^2 such pairs, by the union bound the probability that the algorithm fails is at most $m^2e^{-\alpha}$. It then remains to observe that if $\alpha \geq 2\log m + \log\frac{1}{\delta}$ then $m^2e^{-\alpha} \leq \delta$.

Fix a pair (c, f), where c is a free candidate whose real parent is p, and $f \in C \setminus \{c, p\}$. We can view the profile V as a random variable. Let $E_{1,V}$ be the event that some vote $v \in V$ ranks c higher than f, and let $E_{2,V}$ be the event that $B_V(f) > B_V(p)$. If $E_{1,V}$ happens then the algorithm succeeds with (c, f) by the construction of the attachment digraph, and if $E_{2,V}$ happens then the algorithm succeeds with (c, f) since cf is not added to A' in line 5. Thus, the algorithm fails only if neither of these events happens.

Algorithm 6 samples a preference profile V, in the process checking if in every vote f is above c, otherwise returning $\emptyset$. If $B_V(f) \leq B_V(p)$ does not hold at the end, $\emptyset$ is returned as well. That is, the algorithm samples elements of V uniformly at random and swaps V for $\emptyset$ if $E_{1,V} \vee E_{2,V}$ holds.

Recall that the Sampling Algorithm determines the votes candidate by candidate from top down, similarly to how votes are processed by Algorithm 6. We can view vote sampling as concurrent with vote processing by the algorithm. Consider a run of Algorithm 6, and suppose in vote v being processed p

	Stars					Balanced Binary Trees					Caterpillars					Paths				
size	5	10	20	40	80	7	15	31	63	127	6	10	14	18	22	6	10	14	18	22
n_1	4	7	9	11	13	18	276	7K	251K		10	54	272	1K	5K	20	314	5K	79K	1M
n_2	7	9	11	14	16	30	438	10K	366K		17	96	479	2K	10K	40	654	10K	168K	3M
n_3	8	11	13	15	17	41	579	12K	470K		23	130	635	3K	13K	59	927	15K	236K	4M
n_4	4	7	9	11	13	5	11	19	28	36	5	9	13	18	24	3	4	4	4	4
n_5	6	9	11	14	16	8	18	26	35	43	8	14	21	30	38	5	6	6	6	6
n_6	8	11	13	15	17	11	22	32	41	49	11	19	29	40	52	7	7	7	7	7

Table 1: n_i values obtained in the experiments.

appears in some position k, while f and c have not appeared yet. Then, for subsequent positions $i = k+1, k+2, \dots$ the algorithm tests whether $v[i] = c$ in lines 12–13, and returns $\emptyset$ if so, until $v[i] = f$. Recall from the Sampling Algorithm that, once p has appeared, the probability of sampling $v[i] = c$ at each position is at least $\frac{1}{m}$. Hence, each time the condition in line 12 is tested, the probability of returning $\emptyset$ in the next step is at least $\frac{1}{m}$.

On the other hand, at the end $B_V(f) \leq B_V(p)$ has to hold, or the algorithm returns $\emptyset$. Let $d(v)$ be the difference of the rankings of p and f in vote v if p is ranked higher than f in v, and 0 otherwise. Note that in each vote the positions of p and f differ by at least 1, so for $B_V(f) \leq B_V(p)$ to be true, $\sum_{v \in V} d(v) \geq m\alpha$ has to hold. Further, line 12 in Algorithm 6 is executed exactly $d(v)$ times for each vote v, so if the algorithm does not output $\emptyset$, it is executed at least $m\alpha$ times. Hence, the probability that Algorithm 6 does not return an empty set is at most $(1 - \frac{1}{m})^{m\alpha} < \frac{1}{e^{\alpha}}$. $\square$

5 Experiments

In this section, we empirically investigate the number of samples from $\mathcal{U}(T)$ needed to identify a tree T, either with certainty (Experiment 1) or with high probability (Experiment 2). Estimations are based on 2000 trials, unless stated otherwise.

In Experiment 1 we record our empirical estimations of how many votes are sufficient to identify T with certainty, in 50%, 85%, 95% of instances respectively; we refer to these quantities as n_1, n_2, and n_3. That is, if we order the trials by the number of votes needed to identify the tree with certainty, from the lowest to the highest, then n_1 is the number of samples required in the 1000-th trial, n_2 is the number of samples required in the 1700-th trial, and n_3 is the number of samples required in the 1900-th trial.

In Experiment 2, we record our empirical estimations of how many votes are sufficient to achieve δ values of 50%, 15%, 5% respectively, as discussed in Section 4.2; we refer to these quantities as n_4, n_5 and n_6.

We perform experiments for the following families of trees:

- Stars of sizes 5, 10, 20, 40 and 80
- Balanced binary trees of sizes 7, 15, 31, 63, 127
- Caterpillars of sizes 6, 10, 14, 18, 22
- Paths of sizes 6, 10, 14, 18, 22

The binary tree of size 127 was only used in Experiment 2, due to the huge number of samples it would require in Experiment 1. For the same reason, in case of the binary tree of size 63 and the path of size 22 we only performed 40 trials instead of 2000 in Experiment 1. The results are presented in Table 1.

For identifying trees with certainty, we know that paths require exponentially many samples, whereas for stars logarithmically many samples suffice (Theorems 7 and 8); this is confirmed by our experiments. Together with experimental results for caterpillars and binary trees, this seems to suggest that the required number of samples is closely related to the diameter of the graph.

In contrast, for the task of guessing the tree with high probability, paths require very few samples, but caterpillars turn out to be quite challenging. This is consistent with the intuition behind Algorithm 5: for the algorithm to guess the real parent of a leaf correctly, this leaf needs to be ranked immediately after its parent in some sampled vote. For stars, there is very little difference between Experiment 1 and Experiment 2 in terms of the number of samples. For balanced binary trees, the number of required samples is again logarithmic in the size of the tree in Experiment 2, but seems to be exponential, or almost exponential in Experiment 1.

6 Conclusions

We have investigated the difficulty of learning a tree that generates a given preference profile, either with certainty or with high probability. Our results indicate that achieving certainty may be very costly in terms of the number of samples required; however, learning the tree with high probability is feasible even for fairly large trees.

Throughout the paper, we assumed that neither the tree itself nor the assignment of candidates to vertices is known in advance. One can ask similar questions if the tree itself is known, but we need to learn the assignment; these questions are interesting even if the underlying graph is a path.

Acknowledgments

This research was supported by the European Research Council Starting Grant ACCORD, Grant Agreement 639945 (Elkind). Part of this work was done while Sliwinski was a student at University of Oxford.

References

[Arrow, 1951] Kenneth J. Arrow. Social choice and individual values. 2nd edition, 1951.

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

[Black, 1948] Duncan Black. On the rationale of group decision-making. *The Journal of Political Economy*, pages 23–34, 1948.

[Brandt *et al.*, 2015] Felix Brandt, Markus Brill, Edith Hemaspaandra, and Lane A. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. *Journal of Artificial Intelligence Research (JAIR)*, 53:439–496, 2015.

[Brandt *et al.*, 2016] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. Introduction to computational social choice. In *Handbook of Computational Social Choice*, pages 1–20. 2016.

[Chamberlin and Courant, 1983] John R. Chamberlin and Paul N. Courant. Representative deliberations and representative decisions: Proportional representation and the Borda rule. *American Political Science Review*, 77(3):718–733, 1983.

[Conitzer, 2009] Vincent Conitzer. Eliciting single-peaked preferences using comparison queries. *Journal of Artificial Intelligence Research*, 35:161–191, 2009.

[Cornaz *et al.*, 2012] Denis Cornaz, Lucie Galand, and Olivier Spanjaard. Bounded single-peaked width and proportional representation. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 270–275, 2012.

[Demange, 1982] Gabrielle Demange. Single-peaked orders on a tree. *Mathematical Social Sciences*, 3(4):389–396, 1982.

[Dey and Misra, 2016] Palash Dey and Neeldhara Misra. Elicitation for preferences single peaked on trees. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 215–221, 2016.

[Faliszewski *et al.*, 2011] Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Information and Computation*, 209(2):89–107, 2011.

[Faliszewski *et al.*, 2017] Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner voting: A new challenge for social choice theory. In U. Endriss, editor, *Trends in Computational Social Choice*. AI Access Foundation, 2017.

[Gibbard, 1973] Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, pages 587–601, 1973.

[Inada, 1964] Ken-ichi Inada. A note on the simple majority decision rule. *Econometrica*, pages 525–531, 1964.

[Inada, 1969] Ken-ichi Inada. The simple majority decision rule. *Econometrica*, pages 490–506, 1969.

[Mirrlees, 1971] James A. Mirrlees. An exploration in the theory of optimum income taxation. *The Review of Economic Studies*, 38(2):175–208, 1971.

[Moulin, 1980] Hervé Moulin. On strategy-proofness and single peakedness. *Public Choice*, 35(4):437–455, 1980.

[Moulin, 1991] Hervé Moulin. *Axioms of cooperative decision making*. Cambridge University Press, 1991.

[Peters and Elkind, 2016] Dominik Peters and Edith Elkind. Preferences single-peaked on nice trees. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages 594–600, 2016.

[Roberts, 1977] Kevin W. S. Roberts. Voting over income tax schedules. *Journal of Public Economics*, 8(3):329–340, 1977.

[Satterthwaite, 1975] Mark Allen Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[Sen and Pattanaik, 1969] Amartya K. Sen and Prasanta K. Pattanaik. Necessary and sufficient conditions for rational choice under majority decision. *Journal of Economic Theory*, 1(2):178–202, 1969.

[Sen, 1966] Amartya K. Sen. A possibility theorem on majority decisions. *Econometrica*, pages 491–499, 1966.

[Skowron *et al.*, 2015] Piotr Skowron, Piotr Faliszewski, and Arkadii M. Slinko. Achieving fully proportional representation: Approximability results. *Artificial Intelligence*, 222:67–103, 2015.

[Trick, 1989] Michael A. Trick. Recognizing single-peaked preferences on a tree. *Mathematical Social Sciences*, 17(3):329–334, 1989.

[Walsh, 2007] Toby Walsh. Uncertainty in preference elicitation and aggregation. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 3–8, 2007.

[Walsh, 2015] Toby Walsh. Generating single peaked votes. *CoRR*, abs/1503.02766, 2015.

[Yu *et al.*, 2013] Lan Yu, Hau Chan, and Edith Elkind. Multiwinner elections under preferences that are single-peaked on a tree. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2013.