Robust Winners and Winner Determination Policies under Candidate Uncertainty

Craig Boutilier, Jérôme Lang, Joel Oren and Héctor Palacios

Abstract

We consider voting situations in which a group considers a set of options or candidates, but where some candidates may turn out to be unavailable. If determining availability is costly (e.g., in terms of money, time, or computation), it may be beneficial for the group to vote prior to determining candidate availability, and only test the winner's availability after the vote. However, since few voting rules are robust to candidate deletion, winner determination usually involves a number of such availability tests. We outline a model for analyzing such problems. We define *robust winners* relative to potential candidate unavailability, a notion that is tightly related to control by candidate addition. Then assuming a distribution over availability and costs for availability tests (or *queries*), we define an *optimal query policy* for a vote profile to be one with minimal expected query cost that determines the true winner. We describe a dynamic programming algorithm for computing optimal query policies, as well as a myopic heuristic approach using *information gain* to choose queries. Finally, we outline a number of theoretical and practical questions raised by our model.

1 Introduction

There are many social choice situations in which members of a group may need to specify their preferences over a set of alternatives or *candidates* without knowing whether any specific candidate is in fact viable or *available* for selection. Lu and Boutilier [12] propose the *unavailable candidate model* for studying situations in which potential winners are approached sequentially and the first available candidate is the winner (e.g., consider a hiring committee deciding on the order in which to make job offers, knowing that candidates may refuse). In this paper, we consider a more flexible situation in which determining candidate availability is *costly*, but does not commit one to selecting the first available candidate as the winner.

In our setting, voting over the set of *potential candidates* prior to determining availability can often make sense. For example, a group of friends deciding on a restaurant may attempt to (perhaps partially) determine their aggregate preferences prior to calling (or walking to) restaurants to find out if reservations are possible. A legislative body deciding amongst various public projects may vote prior to knowing their precise financial costs, since the process of budgeting—with engineering estimates, environmental impact studies, etc.—is itself costly. In an AI planning context, a group may vote on the collection of goals to pursue prior to knowing their feasibility, since determining the feasibility of any specific goal set involves solving a computationally difficult problem. In each of these cases, having some idea of who might potentially win an election can narrow the set of *availability tests* that need to be performed, hence the financial, time or computational cost of determining the true winner. But since almost no practical voting rules are robust to candidate deletion, reliably declaring a winner requires making *some* availability tests.

In this paper, we describe a model for addressing such problems, identify a number of key concepts and interesting computational questions that arise in this model, and make some first steps toward solving them. In rough outline, our model assume a set of *potential candidates* X, voter preferences over X in the form of a vote profile $\mathbf{v}$, and some voting rule r. Since candidate availability is uncertain, we assume some distribution P over subsets of X, where $P(A)$ is the probability of A being the true available set, and we assume that each candidate x can be *queried*, for a cost, to determine its availability. Our aim is to determine the winner $r(\mathbf{v}^{\downarrow A})$ of the election (where $\mathbf{v}^{\downarrow A}$

is the restriction of $\mathbf{v}$ to available candidates) in a way that minimizes expected query cost (e.g., number of phone calls, planning problems to be solved, etc.).

To this end, we are primarily interested in *query policies* that propose a (conditional) sequence, or tree, of queries designed to determine enough information about the available set A to declare a winner. Notice that we *need not know A precisely* to determine the winner of an election. Given responses to some set of queries, we know that some candidates $Q^+ \subseteq X$ are available, and some $Q^- \subseteq X$ are not. Relative to such an *information set* $\langle Q^+, Q^- \rangle$, we say x is a *robust winner* if $r(\mathbf{v}^{\downarrow A}) = x$ for any $Q^+ \subseteq A \subseteq X \setminus Q^-$. In other words, this information set is sufficient to determine the winner regardless of the availability of the remaining candidates (e.g., if x is a *majority* winner in a plurality election, the status of other candidates is irrelevant if x is available). The problem of determining a robust winner has very tight connections to the problem of control by candidate addition [2], as we discuss in Sec. 3.

Query policies need only ask enough queries to determine a robust winner. While computing robust winners can be computationally difficult for some voting rules, our primary concern is minimizing query costs, which are much more important than winner determination costs. In Sec. 4 we formulate the query problem as one of constructing a minimal cost *decision tree* over features corresponding to the availability of specific candidates, and whose goal is to classify available sets A according to their winners. We describe a (relatively) inexpensive dynamic programming algorithm for computing optimal query policies; but we also explore the use of standard decision tree induction methods based on *information gain* [16] which are much more computationally tractable. We also consider policies tuned to extreme availability probabilities. Finally, in Sec. 5 we outline a number of interesting theoretical questions, some of which whose answers can have further practical impact on the construction of optimal query policies.

2 Voting with Uncertain Candidate Availability

We first outline our model for winner determination with uncertainty in candidate availability and briefly discuss relevant related work.

2.1 The Model and Decision Problem

We assume a set of n voters N and a set of m *potential candidates* X, with each voter $i \in N$ having a complete, strict preference ordering or *vote* v_i over X, with *vote profile* $\mathbf{v}$ denoting the vector of all votes. A *voting rule* r maps every profile to a (unique) winning candidate (we assume ties are broken in some fashion). We consider rules such as plurality, Borda, and Copeland below, but our framework is completely general. Given a profile $\mathbf{v}$, let $m(\mathbf{v})$ be its majority graph, and $m(\mathbf{v})^*$ the transitive closure of $m(\mathbf{v})$. To make things simpler, we assume an odd number of voters so that $m(\mathbf{v})$ is a tournament. The *top cycle* $TC(\mathbf{v})$ of $\mathbf{v}$ is the set of all candidates x such that for all $y \neq x$, $(x, y) \in m(\mathbf{v})^*$. TC (plus a tie-breaking mechanism) is also considered as a voting rule. A voting rule r is *Smith-consistent* if $r(\mathbf{v}) \in TC(\mathbf{v})$ for any profile $\mathbf{v}$. We assume that r can be applied directly to profiles over arbitrary subsets of X: we use $\mathbf{v}^{\downarrow A}$ to denote the restriction of $\mathbf{v}$ to candidates A, obtained by deleting elements of $X \setminus A$ from each vote, and $r(\mathbf{v}^{\downarrow A})$ to denote the winner w.r.t. this restricted profile. We sometimes use the notation $w(A)$ to denote this winner, suppressing mention of r and $\mathbf{v}$.

We now turn our attention to the possibility that certain candidates in X may be unavailable. There are two natural ways to address this. First, we might first check the availability of all candidates in X, and elicit votes over the set of available candidates $A \subseteq X$. This has the advantage of minimizing vote elicitation costs: voters need not rank or compare unavailable candidates. However, if testing the availability of candidates is itself costly, as discussed above, this may make far more availability tests than needed given voter preferences.

A second approach is to first elicit voter rankings over the entire set X, then use this information to focus attention on "relevant" availability tests. This has the advantage of reducing the cost of availability tests. This is most appropriate when such tests are costly relative to preference assessment or elicitation, as in our examples above. Determining suitable tests is, nonetheless, far from straightforward. One obvious approach is to use the voting rule r to rank candidates, test availability in the order of this aggregate ranking, and select the first available candidate as the winner. However, this assumes that the choice function implemented by voting rule r is *rationalizable*, which is rarely the case. Consider the profile $\mathbf{v}$ with 4 votes abc, 3 votes bca and 2 votes cab. Ranking candidates by their plurality score gives aggregate ranking abc. The policy above, once learning a is available, would select a as the winner without further tests; but if we were to learn that b is unavailable and c available, the true plurality winner for $\mathbf{v}^{\downarrow\{a,c\}}$ is c. This paradox arises since, under mild conditions, no non-dictatorial voting rule is robust to the deletion of non-winning alternatives [4]. As a result, choosing a winner usually requires confirming the availability of specific candidates.[1] Furthermore, minimizing the costs of such availability tests is non-trivial.

We model candidate availability as follows: we partition X into a (possibly empty) *known set* $Y \subseteq X$ of candidates that are sure to be available, and an *unknown set* $U = X \setminus Y$ for which availability is uncertain.[2] Let $\mathcal{A}$ denote the family of subsets $Y \subseteq A \subseteq X$, where $A \in \mathcal{A}$ is a possible *available set*. The general unavailable candidate model [12] requires a distribution P over 2^U, where $P(S)$ denotes the probability that $S \subseteq U$ is the actual available set of candidates from those in U. We assume for simplicity that the availability of each candidate $x \in U$ is independent, given by probability p_x. This induces the obvious distribution over the available sets $\mathcal{A}$.

For any $x \in U$, we assume one can *query* x using some *availability test* to determine its availability (e.g., calling for a restaurant reservation, computing plan for some goal x), which incurs a cost c_x. Informally, a *query policy* (see Sec. 4) consists of a tree whose interior nodes are labeled by queries, edges by availability, and leaves by winners. We desire policies that, given a profile $\mathbf{v}$, accurately determine the winner w.r.t. the *actual* available set A with minimum expected query cost. After any sequence of queries and responses, we have refined information about the available set: an *information set* is an ordered pair $Q = \langle Q^+, Q^- \rangle$, where $Q^+ \subseteq U$, $Q^- \subseteq U$, and $Q^+ \cap Q^- = \emptyset$; intuitively, Q^+ (resp. Q^-) is the set of queries (candidates) for which positive (resp. negative) availability has been determined.

Clearly, winners can often be determined without full knowledge of candidate availability. In our example above, knowing that a and b are available suffices to declare a the winner: availability of c is irrelevant; knowing a is available and c is unavailable is also sufficient to select a.

Definition 1 *Let r be a voting rule, $\mathbf{v}$ a profile over candidate set X, and $Y \subseteq X$ a set of candidates known to be available. We say that $x \in Y$ is a* robust winner *w.r.t.* $\langle X, Y, \mathbf{v}, r \rangle$ *if, for any A such that $Y \subseteq A \subseteq X$, $r(\mathbf{v}(A)) = x$.*

Intuitively, a winner is robust if it not only wins in the original profile $\mathbf{v}$, but continues to win no matter which candidates in $X \setminus Y$ are deleted. In our setting, the existence of a robust winner relative to the current information set is necessary and sufficient to stop any querying process. Specifically, we say that information set Q is *r-sufficient* if there is a robust winner x w.r.t. $\langle X \setminus Q^-, Y \cup Q^+, \mathbf{v}, r \rangle$. For any r-sufficient information set Q, let $w(Q)$ denote this (unique) robust winner.

2.2 Related Work

Lu and Boutilier [12] study a setting where the set of available candidates is unknown at the time of voting, and assume a distribution over available sets A as we do. Unlike our model, they assume a's availability cannot be tested without "offering it the win," hence focus on computing optimal *ranking*

[1]At a minimum, one might require that the winner itself be available, but we consider exceptions to this below.

[2]If any candidates are known to be unavailable *a priori*, we remove them from set X.

policies, as discussed above (see also Baldiga and Green [1] who develop a related availability model with different motivations). Wojtas and Faliszewski [19] also consider candidates with uncertain availability in a counting version of *control by adding candidates* (see below), a problem that is closely related to ours. Their input consists of a subset of candidates Y known to be available, a distribution over subsets of $X \setminus Y$, and votes over X; given a fixed voting rule, they consider the complexity of computing the probability that a given candidate is the winner.

Considerable recent research, starting with Chevaleyre *et al.* [3], studies a variant of the possible winner problem where the candidate set is not known at the time of voting, but take the opposite perspective to ours. A *lower bound* on the candidate set is known initially, and new candidates may join after preferences for the initial set have been elicited. Efficient *preference elicitation* (as opposed to availability testing) protocols are developed to identify the winner.

Our model is also strongly related to *control via adding candidates* [2, 10], in which the initial candidates can be augmented by set of "spoiler" candidates, and a chair, with perfect knowledge of the votes, attempts to find a subset of spoiler candidates whose addition makes her preferred candidate win. We develop the connections to our model further in Sec. 3. Rastegari *et al.* [17] also look for optimal knowledge-gathering policies in a social choice context, although their setting and assumptions differ from ours. Their goal is to find a stable matching (e.g., between companies and job applicants), and knowledge-gathering actions (e.g., interviews of applicants) are intended to reduce uncertainty in the agents' rankings (e.g., a company's assessment of an applicant).

3 Computing Robust Winners

We first consider the problem of identifying or verifying robust winners given some available set. If x is a robust winner w.r.t. $\langle X, Y, \mathbf{v}, r\rangle$, it must meet two obvious necessary conditions: $x \in Y$ and $x = r(\mathbf{v})$ (obtained by setting $A = Y$ and $A = X$). We have the following key result:

Proposition 1 *Let r be a voting rule, $\mathbf{v}$ a profile over X, and $Y \subseteq X$ a set of available candidates. Candidate x is a robust winner w.r.t. $\langle X, Y, \mathbf{v}, r\rangle$ iff there is no destructive control against x by adding candidates, where the spoiler set is $U = X \setminus Y$.*

The proof is immediate: the chair has a destructive control against x via adding candidates iff there is a spoiler set $S \subseteq U$ such that $r(\mathbf{v}^{\downarrow Y \cup S}) \neq x$ (i.e., x is not a robust winner). Since the robust winner problem is equivalent to the complement of the problem of DESTRUCTIVE CONTROL BY ADDING CANDIDATES, results in the literature on election control directly determine the complexity of checking the existence of robust winners for several voting rules: it is coNP-complete for plurality [2, 10], Bucklin [5] and ranked pairs [15]; and it is polynomial for Copeland [6] and maximin [7]. The latter two results come with efficient algorithms for the robust winner problem. These results suggest that the problem tends to be simpler for voting rules that are based on the (unweighted or weighted) majority graph, because majority preference between two candidates x, y does not depend on the availability of other candidates. We provide a simple characterization of robust winners for the top cycle rule (recall that we assume n odd):

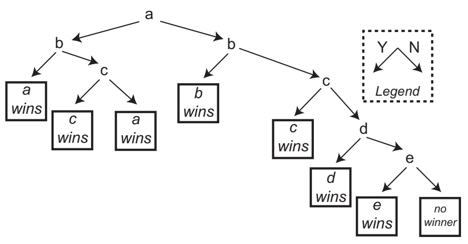
Proposition 2 *Let r be the top cycle rule. Candidate x is a robust winner w.r.t. $\langle X, Y, \mathbf{v}, r\rangle$ iff, for all $y \in X$, there is a path from x to y in $m(\mathbf{v})$ that goes only through candidates in Y.*[3]

As a corollary, checking whether x is a robust winner can be done in polynomial time.

Another interesting notion is that of an *irrelevant candidate*, which can be exploited in computing both robustness and optimal query policies (Sec. 4).

Definition 2 *Let $\mathbf{v}$ be a profile over X, $x \in X$, $Y \subseteq X \setminus \{x\}$ be the known available candidates, and r a voting rule. Candidate x is* irrelevant *w.r.t. $\langle \mathbf{v}, Y, r\rangle$ if for any $A \subseteq X \setminus (Y \cup \{x\})$, we have $r(\mathbf{v}^{\downarrow Y \cup A \cup \{x\}}) = r(\mathbf{v}^{\downarrow Y \cup A})$.*

[3]Proofs of results omitted for space reasons can be found in a longer version of the paper.



Vote profile:

$abcde$	$abcde$	$bcaed$
$bcead$	$adbec$	$cdeba$
$adcbe$	$cbade$	$cdbea$

Figure 1: A plurality-sufficient query policy for vote profile shown.

Notice that if x is irrelevant for Y, we need not consider the availability of x when testing the robustness of any candidate in $X \setminus (Y \cup \{x\})$ w.r.t. Y (or any superset of Y). Similarly, in any policy for determining winners, an availability test for an irrelevant x is useless once Y (or any superset) is known to be available, a fact we can exploit below. We have the following simple characterization of irrelevant candidates for a rich class of voting rules—informally, it says that once we know that at least one candidate in the top cycle is available, removing any candidate that is not in the top cycle cannot impact the choice of winner.

Proposition 3 *Let r be a voting rule satisfying the following property: for any profile $\mathbf{v}$, if $x \notin TC(\mathbf{v})$ then $r(\mathbf{v}^{\downarrow X\setminus\{x\}}) = r(\mathbf{v})$. For any $\mathbf{v}$, any $Y \subseteq X$ s.t. $Y \cap TC(\mathbf{v}) \neq \emptyset$, and any $x \in X \setminus TC(\mathbf{v})$, x is irrelevant w.r.t. $\langle \mathbf{v}, Y, r\rangle$.*

Prop. 3 applies, in particular, to the top cycle, Copeland, Slater and Banks rules.

4 Minimizing Expected Query Cost

We now consider the problem of computing optimal query policies that determine enough information about candidate availability to be able to declare a (robust) winner. We formulate the problem as one of *cost-sensitive decision tree construction*, present some experiments comparing the performance of myopic heuristics to optimal dynamic programming, and discuss a number of interesting theoretical and practical directions for future research.

4.1 Optimal Query Policies

A *query policy* is a binary decision tree T in which each non-leaf node n is labeled by a query $q(n) \in U$, the two outgoing edges from non-leaf node n are labeled by responses *yes* (or "available") and *no* (or "unavailable"), and each leaf l is labeled by an element $w(l)$ of X (the proposed *winner* given the query-response path to l). Let $yes(n)$ and $no(n)$ denote the yes/no successors of node n in T. Any path from the root of T to a leaf l induces the obvious information set $Q(l) = \langle Q^+(l), Q^-(l)\rangle$. A policy/tree T is r-*sufficient* w.r.t. $\mathbf{v}$ if: (a) the information set $Q(l)$ for each leaf l is r-sufficient; and (b) each leaf l is labeled with the robust winner $w(Q(l))$. For any $A \in \mathcal{A}$, let $l(A)$ denote the (unique) leaf that will be reached when responses are dictated by A, and $\pi(A)$ the corresponding path. Fig. 1 shows a vote profile and an r-sufficient tree for plurality voting.

The *query cost* of a (complete) path π in T is $c(\pi) = \sum_{x\in\pi} c(x)$, i.e., the sum of the costs of the queries on π. The *expected query cost* $c(T)$ of policy T is simply $E_{A\sim P}c(\pi(A))$. This can be computed in bottom up fashion as follows, with $c(T)$ being the cost of the root of T.

$$\begin{aligned} c(l) &= 0 \text{ for leaf } l; \\ c(n) &= c_{q(n)} + p_{q(n)} c(yes(n)) + (1 - p_{q(n)}) c(no(n)) \text{ for non-leaf } n. \end{aligned}$$

If all candidates are available with probability $p = 0.9$, the tree in Fig. 1 has an expected query cost of 2.10. Our aim is to find a minimal cost, r-sufficient (i.e., optimal) policy T:

$$\arg\min\{c(T) : T \text{ is } r\text{-sufficient for } \mathbf{v}\}.$$

The problem of computing a minimal cost policy can be cast as a standard decision tree construction (or induction) problem [16]. We can view every available set $A \in \mathcal{A}$ as a training example labeled with its winner $w(A)$. We encode A using a binary feature vector $f(A)$, where each feature corresponds to the presence or absence of a specific candidate in U, and label it with associated probability $P(A)$. Thus our initial set of training examples is simply

$$\{\langle A, w(A), P(A)\rangle : A \in \mathcal{A}\},$$

where each A is encoded by a binary feature vector of length $|U|$.

Clearly the set of training examples has exponential size, requiring winner computation for exponentially many "elections" (of various sizes). Even if winner determination for a fixed candidate set is easy (i.e., polynomial time) for the voting rule in question, simply constructing the inputs for DP (or our myopic decision tree approach to follow) will be difficult.[4] We discuss ways to prune the set of training examples below. Cost-optimal decision trees can be computed readily using dynamic programming (DP) [8] or branch-and-bound [13]. For general binary classification, the (decision variant of) the problem of computing optimal decision trees is NP-complete, even with uniform probabilities and costs [11].[5] However, given the importance of minimizing query costs, we believe even intense computation will be worthwhile.

Standard DP for decision trees structures the problem by considering *sets of training examples* $E \subseteq \mathcal{E} = 2^{\mathcal{A}}$. A set E is *pure* if all examples in E are labeled with the same winner. The *optimal cost function* c^* for an arbitrary example set $E \subseteq \mathcal{E}$ is the cost of the optimal policy knowing initially that the available set A lies within E:

$$c^*(E) = \begin{cases} 0 & \text{if } E \text{ is pure} \\ \min_{q\in U} p(q^+|E)c^*(E_q^+) + p(q^-|E)c^*(E_q^-) + c_q & \text{if } E \text{ is not pure} \end{cases}$$

Since $c^*(E)$ depends only on the optimal optimal costs for subsets of E, DP can be used, computing optimal costs for smaller sets before larger ones.

Naïve DP is doubly exponential in the size of the candidate set U: $|\mathcal{E}| = 2^{|\mathcal{A}|} = 2^{2^{|U|}}$. But the structure in our problem gives us very restricted subsets of training examples. First, observe that example set E_0 at the root consists of all subsets A (i.e., $E_0 = \mathcal{A}$). Second, every query-response path of length k gives a set E equal to E_0 restricted to the instantiation of k queries, i.e., the set of feature vectors where k positions are fixed to some k-bit vector, and all $2^{|U|-k}$ instantiations of the remaining $|U| - k$ features are present. Thus the only realizable sets E have size $2^{|U|-k}$, for some $0 \le k \le |U|$, and each such E has the form described above. Hence, the number of "reachable" example sets required for DP is exponential rather than doubly exponential:

$$\sum_{k=0}^{|U|} 2^k \binom{|U|}{k} = 3^{|U|}.$$

Thus the optimal query policy can be computed in $O(3^{|U|})$ time for any profile and any voting rule.

The DP approach is generic and exploits no structure at all in the profile, nor any special properties of the voting rule itself. An important research direction is the refinement of the DP algorithm for specific voting rules in a way that could (perhaps drastically) prune the subsets A that need to be explicitly considered. For example, for rules such as top cycle, Copeland, Slater and Banks, Prop. 3

[4]Indeed, it will be NP-hard in general; we thank one of the reviewers for this observation.

[5]We are exploring reductions to confirm the complexity of our problem, which we believe to be PSPACE-complete.

allows us to "collapse" potentially large numbers of candidate subsets—those that vary only in the availability of "irrelevant candidates"—and treat them as a single training example with a unique winner. Similarly, rules that satisfy the "majority winner property"—i.e., if a candidate is in the first position of the majority of votes, it must win—allows significant pruning as well: in any subset Y with such a majority winner x, x's availability renders all remaining candidates irrelevant. This type of pruning has two direct benefits: it reduces both the (explicit) set $\mathcal{A}$ (i.e., number of training examples), and the number of subsets E of $\mathcal{A}$ that must be considered (as we discuss below).

4.2 Myopic Query Tree Construction

Because of the NP-hardness of optimal decision tree construction in general, and the exponential complexity of DP in particular, the machine learning community has considered heuristic, *myopic* approaches to decision tree induction, one of the most popular being C4.5, which is based on *information gain* [16]. Extensions to cost-sensitive classification have been considered as well [9, 18], and such schemes can be adapted to our setting easily. We now outline such an approach.

For any set of training examples $E \subseteq \mathcal{E}$, define $w(E)$ to be the set of winners $w(A)$ that occur in some example $A \in E$. Let $p_E(x) = \sum\{P(A) : A \in E, w(A) = x\}$ be the probability that x wins in training set E. The *entropy* of E is

$$H(E) = \sum_{x \in w(E)} -p_E(x) \log p_E(x).$$

A *split* of E on a feature (i.e., candidate, query) q partitions E into those examples $E_q^+ \subseteq E$ where q is available, and $E_q^- \subseteq E$ where q is not. The *conditional entropy* of training set E given q is:

$$H(E|q) = p(q^+|E)H(E_q^+) + p(q^-|E)H(E_q^+).$$

We define the *information gain* associated with query q to be:

$$IG(E|q) = H(E) - H(E|q).$$

Myopic decision tree construction proceeds as follows. We first initialize the tree with a single leaf (root) node n_0 and associate with n_0 the set $E_0 = \mathcal{A}$ all labeled training examples. Then we repeat the following operations on unprocessed nodes until no nodes are unprocessed. Let n be an unprocessed node and $E(n)$ be its associated training set:

1. If n is pure, designate it processed; label it with its (unique) winner (it now becomes a leaf).
2. If n is not pure, then:
   (a) For each (non-redundant) feature split $q \in U$, compute its information gain $IG(E(n)|q)$.
   (b) Split n using the query q with maximum information gain, creating children: $n_q^+ = \mathit{yes}(n)$ on the yes edge, associated with examples E_q^+; and $n_q^- = \mathit{no}(n)$ on the no edge, associated with examples E_q^-.
   (c) Designate n processed, and n_q^+ and n_q^- unprocessed.

Processing of any node is linear in the number of training examples at a node, hence the complexity of myopic decision tree induction is linear in (a possibly pruned) $\mathcal{A}$ and the size of the resulting tree. Since $\mathcal{A}$ has size $2^{|U|}$ in the worst case (if unpruned), complexity of myopic induction is $O(2^{|U|})$, i.e., significantly more efficient than optimal tree construction using DP (which has unpruned complexity $O(3^{|U|})$). This improved efficiency comes at a price, since the use of information gain is not guaranteed to result in a policy with minimal expected query cost. However, we expect it to work well in practice; and it *is guaranteed* to provide a *correct* policy (i.e., one that determines the true winner).

If we are willing to live with potential error in the declaration of the winner, other forms of approximation can be considered in constructing the policy. We describe two here, but leave their detailed investigation to future research. The approaches above ensure we always obtain the correct winner, since the policy tree has only pure leaves (i.e., all candidate sets A at a leaf have the same winner). One form of approximation is to terminate the querying process at *impure leaves*, requiring only that we be "sure enough" about the identity of the winner, and predicting a winner despite this residual uncertainty. This is analogous to cost-sensitive classification [9, 18], where both tests and and *prediction errors* have costs. In our setting, we have two distinct types of misclassification errors/costs: (a) if we predict/choose a winner who turns out to be unavailable; (b) if we predict/choose a winner that is available, but is not the true winner given the actual (unknown) available set. In general, we expect the former to be much worse than the latter.[6] An important research direction will be to to modify the query policy algorithms so these costs are taken into account.

Another important approximation, which can help reduce the number of training examples and render myopic decision tree construction fully tractable, is to build the tree using *sampled availability sets*, where training examples A are drawn from the distribution P over $\mathcal{A}$. With a constant number of sampled sets, decision tree induction becomes linear in the number of candidates and size of T. Sample complexity results then become an important research direction [9].

4.3 Extreme Availability Probabilities

When candidate availability probabilities are extreme, i.e., close to close to 1 or 0, constructing optimal query policies becomes much easier. Assume $p_x = p$ for all x and all query costs are identical. We first consider the case where p is very close to 1, reflecting settings where an unavailable candidate would be exceptional. Let $p = 1 - O(\varepsilon)$. Consider the following query policy $\mathit{Extreme}(\mathbf{v})$, which (informally) proceeds as follows. We initialize the *potential set* X to contain all candidates, the *known set* $Y = \emptyset$, and the *current winner* $w = r(\mathbf{v})$. We then repeat the following steps:

1. look for a minimum-cardinality subset Z of $X \setminus Y$ such that w is a robust winner for $Y \cup Z$ (note that Z must contain w if w is not known to be available);
2. check the availability of all candidates in Z;
3. if all candidates in Z are available, stop and output w;
4. if not, remove the unavailable candidates from the profile $\mathbf{v}$, recompute $w = r(\mathbf{v})$, and go back to step 1.

As an example, let r be plurality and consider the profile $\mathbf{v}$ shown in Fig. 1. The (initial) current winner $r(\mathbf{v})$ is a. We then have $Z = \{a, b\}$. Under our extreme availability assumption, with high probability we will learn that both a and b are available, in which case we stop and output a. However, suppose that we learn a is available but b is not. We would then replace $\mathbf{v}$ by $\mathbf{v}^{\downarrow\{a,c,d,e\}}$, giving a new current winner c, and $Z = \{c\}$. After checking Z, suppose we learn c is not available; we then replace $\mathbf{v}$ by $\mathbf{v}^{\downarrow\{a,d,e\}}$, giving new current winner a, and $Z = \emptyset$: we stop and output a.

It is not hard to show that $\mathit{Extreme}(\mathbf{v})$ terminates, and returns the true winner $r(\mathbf{v}^{\downarrow A})$ for the actual available set A, provided at least one candidate is actually available. Now suppose $Y \subseteq X$ is a smallest (cardinality) set of candidates such that $r(\mathbf{v})$ is a robust winner for Y, and $|Y| = k$. Then we have that the expected cost of the policy is $k + O(\varepsilon)$. We can also show that any query policy must have this expected cost:

[6]For instance, going to a restaurant without confirming availability, then arriving to find out it has no space, is worse than going to that same restaurant having obtained a reservation and then finding out another restaurant might have been preferable because some others became unavailable.

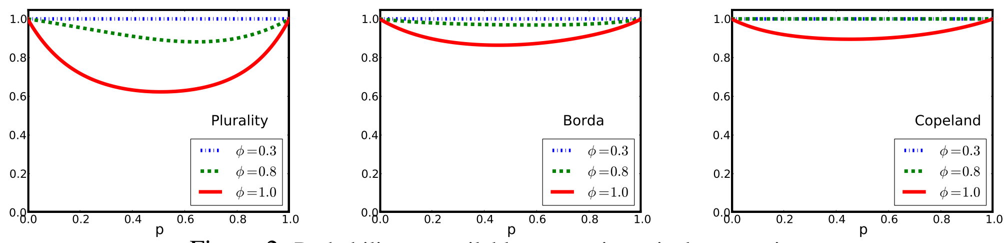


Figure 2: Probability an available naïve winner is the true winner.

Lemma 1 *Any r-sufficient query policy has an expected query cost of at least $k - O(\varepsilon)$.*

Proof: With probability $p^{|X|}$, all candidates are available. When all candidates are available, $x = r(\mathbf{v})$ is the true winner, and to verify it we must check the availability of $k-1$ other candidates (this being the smallest size set that admits a robust winner). Therefore, the expected cost of any query policy is at least $p^{|X|} \cdot k = k - O(\varepsilon)$. ■

These facts together directly demonstrate the following:

Proposition 4 *The policy $Extreme(\cdot)$ is asymptotically optimal as $\varepsilon \to 0$.*

The opposite case, where all candidates are highly unlikely to be available, is much less interesting: it is very likely that we must check the availability of all candidates—even then it is highly likely that no candidate is available—and run the voting on the available set. The naïve policy that tests all candidates in advance of voting is (asymptotically) optimal.

4.4 Empirical Evaluation

We now describe some simple experiments designed to test the effectiveness of our algorithms for computing query policies, and examine the expected costs of these policies for various voting rules, preference distributions and availability distributions. In all of the experiments that follow, we generate vote profiles using *Mallows distributions* over rankings [14]. The Mallows model is a distribution over rankings given by a modal ranking σ over X and dispersion $\phi \in (0,1]$: the probability of vote v is $\Pr(v|\sigma,\phi) \propto \phi^{d(r,\sigma)}$, where d is Kendall's τ-distance (each vote is drawn independently). Smaller ϕ concentrates mass around σ while $\phi = 1$ gives the uniform distribution, i.e., Mallows with $\phi = 1$ corresponds precisely to the *impartial culture assumption*. In all experiments, we consider $m = 10$ candidates and $n = 100$ voters. We generate vote profiles for $\phi \in \{0.3, 0.8, 1.0\}$, and consider three different voting rules: plurality, Borda and Copeland. We consider various availability probabilities p, which vary depending on the circumstance, but in all cases each candidate has the same availability probability. Results for each problem instance (combination of voting rule, ϕ, p combination) are reported as averages (and other statistics) over 25 randomly drawn vote profiles.

Before exploring the performance of query policies, we first measure the probability of error (i.e., selecting an incorrect winner) associated with a naïve policy which simply selects the *naïve winner*, $r(\mathbf{v})$, without regard to candidate unavailability. Obviously, a lower bound on this error is $1-p$, since the winner will be unavailable with that probability. Fig. 2 shows this error probability *conditional on the winner being available* for the three voting rules considered, for different ϕ, as we vary the availability probability p. For p near 1, the naïve winner is, of course, almost always correct. At the other extreme, when candidates are usually not available, the naïve winner is usually correct also, since it is highly likely to be the only available option. We see that when preferences are very peaked ($\phi = 0.3$), candidate deletion has little impact, since most voters rank all candidates very similarly; but as preferences become more uniform—in particular, for impartial culture ($\phi = 1$)—the impact is dramatic, especially for plurality, and somewhat less so for Copeland. This suggests that testing availability is important even for reasonably high availability probabilities. It is important to

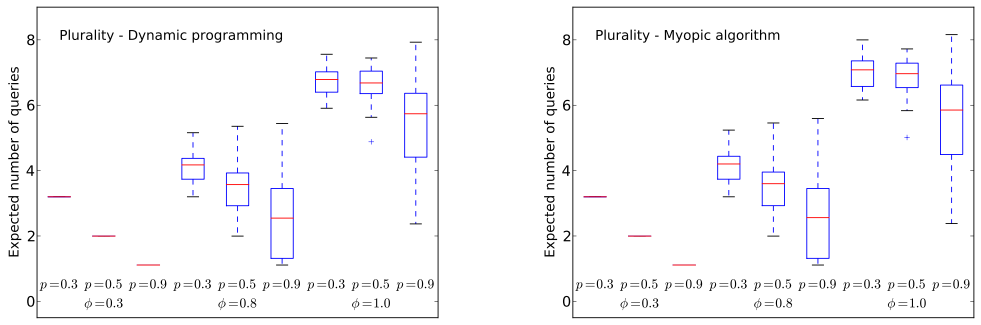


Figure 3: Expected query cost of decision trees for plurality using (a) dynamic programming and (b) the information gain heuristic.

ϕ	0.3			0.8			1.0		
p	0.3	0.5	0.9	0.3	0.5	0.9	0.3	0.5	0.9
Plur-DP	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	4.1 (3.2,5.2)	3.4 (2.0,5.4)	2.7 (1.1,5.4)	6.7 (5.9,7.6)	6.6 (4.9,7.4)	5.4 (2.4,7.9)
Borda-DP	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	3.7 (3.2,4.5)	2.7 (2.0,3.9)	1.7 (1.1,5.0)	5.4 (4.4,6.7)	4.8 (3.2,6.4)	3.3 (1.2,6.2)
Cope-DP	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	3.2 (3.2,3.6)	2.0 (2.0,2.5)	1.1 (1.1,1.3)	4.6 (3.4,5.9)	3.6 (2.1,5.6)	2.2 (1.1,4.5)
Plur-IG	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	4.1 (3.2,5.2)	3.5 (2.0,5.5)	2.8 (1.1,5.6)	7.0 (6.2,8.0)	6.9 (5.0,7.7)	5.6 (2.4,8.2)
Borda-IG	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	3.7 (3.2,4.6)	2.7 (2.0,3.9)	1.7 (1.1,5.0)	5.5 (4.5,7.0)	4.9 (3.2,6.7)	3.3 (1.2,6.2)
Cope-IG	3.2 (3.2,3.2)	2.0 (2.0,2.0)	1.1 (1.1,1.1)	3.2 (3.2,3.6)	2.0 (2.0,2.5)	1.1 (1.1,1.3)	4.7 (3.5,6.7)	3.7 (2.1,5.9)	2.2 (1.1,4.5)

Table 1: Avg. query cost (min, max) for optimal (DP) and myopic (IG) query policies.

realize that these plots give only a sense of the "degree of robustness" of a winner *who is assumed to be available*, even for low p (where the odds of winner availability is low): as such, they do not necessarily provide insight into the value of intelligent availability testing.

We now consider the expected number of queries required to determine the winner under the scenarios described above (using the different values of ϕ) under three different availability probabilities: $p = 0.3, 0.5, 0.9$. Fig. 3 plots expected query cost for plurality, using both dynamic programming (which gives the optimal policy) and the myopic information gain heuristic, for all nine parameter settings. The plots show average expected cost, standard error, and maximum and minimum expected costs over 25 trials. The results for all three rules are shown in Table 1.

The first thing to notice is that, in most settings, the use of optimal query policies can offer significant savings in availability tests relative to the standard approach of first testing the availability of all ten candidates. We see that the myopic heuristic tends to produce trees with costs very close to the optimum: even in the problems instances with the largest average gap (i.e., for plurality with $\phi = 1$), the myopically constructed trees have an expected cost of only 0.3 more queries than optimal on average; in most cases, these trees are almost identical to the corresponding optimum. This suggests that the more efficient myopic approaches will work well at minimizing availability query costs in practice. Not surprisingly, we see strong (negative) correlations between query costs and availability probability in all three voting rules. The query cost is also correlated with dispersion ϕ: when ϕ is greater (more uniform) query costs are higher since preferences are more diverse. When dispersion ϕ is low, given a fixed p, expected query cost is the essentially the same for each voting rule, and the myopic algorithm produces virtually optimal policies (indeed, identical in cost to the optimum up to the reported precision).

Table 2 shows the sizes of the decision trees that result when running both of our algorithms: tree size is only indirectly related to expected query cost, since the relative balance of the trees also impacts expected query costs. Nonetheless we see an expected correlation, though we notice that plurality tends to result in larger trees, especially for $\phi = 1$. We also see that the myopic trees are not significantly larger than the optimal trees, though the differences in size are somewhat more pronounced than the differences in expected query cost described above.

ϕ	0.3			0.8			1.0		
p	0.3	0.5	0.9	0.3	0.5	0.9	0.3	0.5	0.9
Plur-DP	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	52.0 (9,128)	49.6 (9,124)	57.0 (9,148)	233.8 (133,311)	221.2 (121,302)	249.6 (136,318)
Borda-DP	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	24.4 (9,61)	24.0 (9,57)	26.2 (9,63)	114.8 (42,209)	110.3 (41,197)	125.1 (44,226)
Cope-DP	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	10.3 (9,19)	10.3 (9,19)	10.3 (9,19)	58.4 (17,161)	57.8 (17,160)	63.1 (17,185)
Plur-IG	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	59.5 (9,140)	55.4 (9,140)	62.2 (9,163)	290.8 (163,379)	258.6 (145,351)	296.9 (171,402)
Borda-IG	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	26.8 (9,63)	24.1 (9,59)	26.5 (9,68)	136.5 (49,264)	117.8 (42,229)	135.1 (46,253)
Cope-IG	9.0 (9,9)	9.0 (9,9)	9.0 (9,9)	10.3 (9,19)	10.3 (9,19)	10.4 (9,20)	67.1 (21,211)	60.8 (18,178)	70.2 (18,213)

Table 2: Avg. tree size (min, max) for optimal (DP) and myopic (IG) (number of query nodes).

4.5 Query Complexity

Apart from optimizing query policies, the theoretical question of both worst-case query (availability test) complexity, and average-case query complexity for specific distributions, is of interest. Here we sketch some partial results just to offer a flavor for the types of questions one might ask in the unavailable candidate model under our availability testing regime.

Worst-case results take the form: given a voting rule r and availability distribution P, what is the greatest (over possible vote profiles $\mathbf{v}$) expected (over availabilities) query cost of the optimal query policy? Distributions where candidates are highly likely to be available allow one to construct profiles where determining the winner requires almost m queries in expectation, for both plurality and Borda. Our constructions partition votes into two sets each with a distinct "winning" candidate in each set with equal scores (and arranging other candidates appropriately "uniformly" so that a precise subset of available candidates is needed to determine which of the two top candidates wins). The hardness of these cases lies in the high concentration of the binomial distribution. In the extreme case where p is very close to 1, for any voting rule the expected query complexity is

$$\min\{|Y| \text{ s.t. } w = r(\mathbf{v}) \text{ is a robust winner w.r.t. } (r, X, Y)\} + O(\varepsilon)$$

(this can be seen from our earlier construction). By contrast, if p is very close to 0, it is highly likely that we must check the availability of every candidate, giving an expected query complexity $m + O(\varepsilon)$.

With regard to expected query complexity, interesting questions arise when considering the expected optimal query cost not for worst-case profiles, but on average for preference or vote profiles drawn from particular distributions, such as impartial culture, various forms of Mallows models or mixtures, distributions over single-peaked preferences, etc., for various voting rules.

5 Future Directions

We have offered a new perspective on voting in the unavailable candidate model, assuming that testing the viability or availability of candidates is costly. We have presented several new concepts, including those of robust winners, irrelevant candidates, and availability/query policies, and provided algorithms for the computation of both optimal and myopic query policies by exploiting connections to decision tree induction. Our experimental results with plurality, Borda and Copeland voting show the value of optimal querying, with significant savings (relative to determining availability of all candidates) realized over a variety of preference and availability distributions.

There are of course a variety of interesting directions to be pursued in this nascent line of research. Many of these have been suggested above, but we summarize some of them here. A critical direction of both practical and theoretical interest is developing efficient methods for pruning the available sets used in policy construction based on specific properties of voting rules. Heuristic methods for constructing policies could be useful; e.g., we might exploit the probability a given candidate will win—as in the control problem addressed in [19]—to determine the next candidate to query. Sample-based procedures, where only some available sets are classified, may also prove to be important in minimizing computational costs. Other representations of winner determination policies (e.g., decision lists or graphs) may be of value. Extensions to policies that "predict" winners, rather than guaranteeing robustness—whether in a *speculative* (choosing winners that may not be available) or *safe* (only selecting known available candidates) fashion—is of great interest. We are exploring some of the query and communication complexity questions mentioned above. Finally, the question of new opportunities for manipulation under this model seem rather intriguing.

Acknowledgments: Thanks to the reviewers for helpful suggestions. This work was supported in part by NSERC, by project ComSoc (ANR-09-BLAN-0305-01), and by MICINN project TIN2011-27652-C03-02.

References

[1] K. Baldiga and J. Green. Assent-maximizing social choice. *Social Choice and Welfare*, online first, 2011.

[2] J. Bartholdi, C. Tovey, and M. Trick. How hard is it to control an election? *Social Choice and Welfare*, 16(8-9):27–40, 1992.

[3] Y. Chevaleyre, J. Lang, N. Maudet and J. Monnot. Possible winners when new candidates are added: the case of scoring rules. *Proceedings of AAAI-10*, 2010.

[4] B. Dutta, M. Jackson and M. Le Breton. Strategic candidacy and voting procedures. *Econometrica*, 69(4):1013–1037, 2001.

[5] G. Erdélyi, M. Fellows, L. Piras and J. Rothe. Control complexity in Bucklin and fallback voting. arXiv 1103.2230, 2011.

[6] P. Faliszewski, E. Hemaspaandra and H. Schnoor. Copeland voting: ties matter. *AAMAS-08*, pp. 983-990, 2008

[7] P. Faliszewski, E. Hemaspaandra and L. Hemaspaandra Multimode control attacks on elections. *JAIR*, 40:305–351, 2011.

[8] M. R. Garey. Optimal binary identification procedures. *SIAM J. Appl. Math.*, 23:173–186, 1972.

[9] R. Greiner, A. J. Grove and D. Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174, 1992.

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

[11] L. Hyafil and R. Rivest Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5:15–17, 1976.

[12] T. Lu and C. Boutilier. The unavailable candidate model: a decision-theoretic view of social choice. *ACM Conference on Electronic Commerce 2010*, pp.263–274, 2010.

[13] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039, 1978.

[14] C. L. Mallows. Non-null ranking models. *Biometrika*, 44:114–130, 1957.

[15] D. Parkes and L. Xia. A complexity-of-strategic-behavior comparison between Schulze's rule and ranked pairs. AAAI-12, to appear, 2012.

[16] J. R. Quinlan. *C45: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[17] B. Rastegari, A. Condon, K. Leyton-Brown and N. Immorlica. Two-Sided matching with partial information. Working paper, 2011.

[18] P. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.

[19] K. Wojtas and P. Faliszewski. Possible winners in noisy elections. AAAI-12, to appear, 2012.

Craig Boutilier
Department of Computer Science
University of Toronto
Email: `cebly@cs.toronto.edu`

Jérôme Lang
LAMSADE
Université Paris-Dauphine
Email: `lang@lamsade.dauphine.fr`

Joel Oren
Department of Computer Science
University of Toronto
Email: `oren@cs.toronto.edu`

Héctor Palacios
Departamento de Informática
Universidad Carlos III de Madrid
Email: `hpalacio@inf.uc3m.es`