Eliciting Single-Peaked Preferences Using Comparison Queries

Vincent Conitzer
Duke University
Department of Computer Science
Durham, NC 27708, USA
conitzer@cs.duke.edu

ABSTRACT

Voting is a general method for aggregating the preferences of multiple agents. Each agent ranks all the possible alternatives, and based on this, an aggregate ranking of the alternatives (or at least a winning alternative) is produced. However, when there are many alternatives, it is impractical to simply ask agents to report their complete preferences. Rather, the agents' preferences, or at least the relevant parts thereof, need to be *elicited*. This is done by asking the agents a (hopefully small) number of simple queries about their preferences, such as *comparison* queries, which ask an agent to compare two of the alternatives. Prior work on preference elicitation in voting has focused on the case of unrestricted preferences. It has been shown that in this setting, it is sometimes necessary to ask each agent (almost) as many queries as would be required to determine an arbitrary ranking of the alternatives. By contrast, in this paper, we focus on single-peaked preferences. We show that such preferences can be elicited using only a linear number of comparison queries, if either the order with respect to which preferences are single-peaked is known, or at least one other agent's complete preferences are known. We also show that using a sublinear number of queries will not suffice. Finally, we present experimental results.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Economics, Theory

Keywords

Computational social choice, voting, preference elicitation, single-peaked preferences

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

1. INTRODUCTION

In multiagent systems, a group of agents often has to make joint decisions even when the agents have conflicting preferences over the alternatives. For example, agents may have different preferences over possible joint plans for the group, allocations of tasks or resources among members of the group, potential representatives (*e.g.* presidential candidates), *etc.* In such settings, it is important to be able to *aggregate* the agents' individual preferences. The result of this aggregation can be a single alternative, corresponding to the group's collective decision, or a complete aggregate (compromise) ranking of all the alternatives (which can be useful, for instance, if some of the alternatives later turn out not to be feasible). The most general framework for aggregating the agents' preferences is to have the agents *vote* over the alternatives. That is, each agent announces a complete ranking of all alternatives (the agent's *vote*), and based on these votes an outcome (*i.e.* a winning alternative or a complete aggregate ranking of all alternatives) is chosen according to some *voting rule*.[1]

One might try to create an aggregate ranking as follows: for given alternatives a and b, if more votes prefer a to b than vice versa (*i.e.* a wins its *pairwise election* against b), then a should be ranked above b in the aggregate ranking. Unfortunately, when the preferences of the agents are unrestricted and there are at least three alternatives, *Condorcet cycles* may occur. A Condorcet cycle is a sequence of alternatives $a_1, a_2, \ldots, a_k$ such that for each $1 \leq i < k$, more agents prefer a_i to a_{i+1} than vice versa, and more agents prefer a_k to a_1 than vice versa. In the presence of a Condorcet cycle, it is impossible to produce an aggregate ranking that is consistent with the outcomes of all pairwise elections. Closely related to this phenomenon are numerous impossibility results that show that every voting rule has significant drawbacks in this general setting. For example, when there are at least three alternatives, Arrow's impossibility theorem [Arrow, 1963] shows that any voting rule for which the relative order of two alternatives in the aggregate ranking is independent of how agents rank alternatives other than

[1]One may argue that this approach is not fully general because it does not allow agents to specify their preferences over *probability distributions* over alternatives. For example, it is impossible to know from an agent's vote whether that agent prefers its second-ranked alternative to a 1/2 - 1/2 probability distribution over its first-ranked and third-ranked alternatives. In principle, this can be addressed by voting over these probability distributions instead, although in practice this is usually not tractable.

978-81-904262-7-5 (RPS)

these two (*i.e.* any rule that satisfies *independence of irrelevant alternatives*) must either be *dictatorial* (*i.e.* the rule simply copies the ranking of a fixed agent, ignoring all other agents) or conflicting with *unanimity* (*i.e.* for some alternatives a and b, the rule sometimes ranks a above b even if all agents prefer b to a). As another example, when there are at least three alternatives, the Gibbard-Satterthwaite theorem [Gibbard, 1973; Satterthwaite, 1975] shows that for any voting rule that is *onto* (for every alternative, there exist votes that would make that alternative win) and nondictatorial, there are instances where an agent is best off casting a vote that does not correspond to the agent's true preferences (*i.e.* the rule is not *strategy-proof*).

1.1 Single-peaked preferences

Fortunately, these difficulties can disappear if the agents' preferences are restricted, *i.e.* they display some structure. The best-known, and arguably most important such restriction is that of *single-peaked preferences* [Black, 1948]. Suppose that the alternatives are ordered on a line, from left to right, representing the alternatives' *positions*. For example, in a political election, a candidate's position on the line may indicate whether she is a left-wing or a right-wing candidate (and how strongly so). As another example, the alternatives may be numerical values: for example, agents may vote over the size of a budget. As yet another example, the alternatives may be locations along a road (for example, if agents are voting over where to construct a building, or where to meet for dinner, *etc.*). We say that an agent's preferences are *single-peaked* with respect to the alternatives' positions if, on each side of the agent's most preferred alternative (the agent's *peak*), the agent prefers alternatives that are closer to its peak. For example, if the set of alternatives is $\{a, b, c, d, e, f\}$, their positions may be represented by $d < b < e < f < a < c$, in which case the vote $f \succ e \succ b \succ a \succ c \succ d$ is single-peaked, but the vote $f \succ e \succ a \succ d \succ c \succ b$ is not (b and d are on the same side of f in the positions, and b is closer to f, so d should not be ranked higher than b if f is the peak). (Throughout, we will assume that all preferences are strict, that is, agents are never indifferent between two alternatives.) Preferences are likely to be single-peaked if the alternatives' positions are of primary importance in determining an agent's preferences. For example, in political elections, if voters' preferences are determined primarily by candidates' proximity to their own stance on the left-to-right spectrum, preferences are likely to be single-peaked. If other factors are also important, such as the perceived amicability of the candidates, then preferences are not necessarily likely to be single-peaked.

Formally, if an agent with single-peaked preferences prefers a_1 to a_2, one of the following must be true:

- a_1 is the agent's peak,
- a_1 and a_2 are on opposite sides of the agent's peak, or
- a_1 is closer to the peak than a_2.

When all agents' preferences are single-peaked (with respect to the same positions for the alternatives), it is known that there can be no Condorcet cycles. If, in addition, we assume that the number of agents is odd, then no pairwise election can result in a tie. Hence, our aggregate ranking can simply correspond to the outcomes of the pairwise elections. In this case, there is also no incentive for an agent to misreport its preferences, since by reporting its preferences truthfully, it will, in each pairwise election, rank the alternative that it prefers higher.

1.2 Preference elicitation

A key difficulty in aggregating the preferences of multiple agents is the *elicitation* of the agents' preferences. In many settings, particularly those with large sets of alternatives, having each agent communicate all of its preferences is impractical. For one, it can take up a large amount of communication bandwidth. Perhaps more importantly, in order for an agent to communicate all of its preferences, it must first *determine* exactly what those preferences are. This can be a complex task, especially when no guidance is provided to the agent as to what the key questions are that it needs to answer to determine its preferences.

An alternative approach is for an *elicitor* to sequentially ask the agents certain natural *queries* about their preferences. For example, the elictor can ask an agent which of two alternatives it prefers (a *comparison query*). Three natural goals for the elicitor are to (1) learn enough about the agents' preferences to determine the winning alternative, (2) learn enough to determine the entire aggregate ranking, and (3) learn each agent's complete preferences. (1) and (2) have the advantage that in general, not all of each agent's preferences need to be determined. For example, for (1), the elicitor does not need to elicit an agent's preferences among alternatives for which we have already determined (from the other agents' preferences) that they have no chance of winning. But even (3) can have significant benefits over not doing any elicitation at all (*i.e.* having each agent communicate all of its preferences on its own). First, the elicitor provides the agent with a systematic way of assessing its preferences: all that the agent needs to do is answer simple queries. Second, and perhaps more importantly, once the elicitor has elicited the preferences of some agents, the elicitor will have some understanding of which preferences are more likely to occur (and, perhaps, some understanding of why this is so). The elicitor can then use this understanding to guide the elicitation of the next agent's preferences, and learn these preferences more rapidly.

In this paper, we will study the elicitation of single-peaked preferences using only comparison queries. We will focus on approach (3), *i.e.* learning each agent's complete preferences. We will study both the setting where the elicitor knows the positions of the alternatives (Section 4), and the setting where the elicitor (at least initially) does not (Section 5). We will assume that preferences are always single-peaked.[2] Our elicitation algorithms completely elicit one agent's preferences before moving on to the next agent (as opposed to going back and forth between agents). This gives the algorithms a nice online property: if agents arrive over time, then we can elicit an agent's preferences when it arrives, after which the agent is free to leave (as opposed to being forced to wait until the arrival of the next agent).

[2]We note that if it is possible that some agent's preferences are not single-peaked, we can always elicit them as if they were, and then *verify* that we have learned them correctly using an additional $m - 1$ comparison queries. This is done by asking the agent whether it prefers (what we think is) its most preferred alternative to (what we think is) its second-most preferred alternative, its second-most preferred alternative to its third-most preferred alternative, *etc.* If this verification step fails, we can use some other method to re-elicit the agent's preferences.

2. RELATED RESEARCH

A significant body of work on preference elicitation in multiagent systems focuses on combinatorial auctions (for an overview of this work, see Sandholm and Boutilier [2006]). Much of this work focuses on approach (1), *i.e.* learning enough about the bidders' valuations to determine the optimal allocation. (Sometimes, additional information must be elicited from the bidders to determine the payments that they should make according to the Clarke [Clarke, 1971], or more generally, a Groves [Groves, 1973], mechanism.) Example elicitation approaches include ascending combinatorial auctions (for an overview, see Parkes [2006]) as well as frameworks in which the auctioneer can ask queries in a more flexible way [Conen and Sandholm, 2001]. A significant amount of the research on preference elicitation in combinatorial auctions is also devoted to approach (3), *i.e.* learning an agent's complete valuation function. In this research, typically valuation functions are assumed to lie in a restricted class, and given this it is shown that an agent's complete valuation function can be elicited using a polynomial number of queries of some kind. Various results of this nature have been obtained by Zinkevich *et al.* [2003], Blum *et al.* [2004], Lahaie and Parkes [2004], and Santi *et al.* [2004].

There has also been some work on elicitation in voting settings (the setting of this paper). All of that work so far has focused on approach (1), eliciting enough information from the agents to determine the winner, without any restriction on the space of possible preferences. Conitzer and Sandholm [2002] studied the complexity of deciding whether enough information has been elicited to declare a winner, as well as the complexity of choosing which votes to elicit given very strong suspicions about how agents will vote. They also studied what additional opportunities for strategic misreporting of preferences elicitation introduces, as well as how to avoid introducing these opportunities. (Strategic misreporting is not a significant concern in the setting of this paper: under the restriction of single-peaked preferences, reporting truthfully is a dominant strategy when agents simultaneously report their complete preferences, and hence responding truthfully to the elicitor's queries is an *ex-post* equilibrium. As such, in this paper we will make no distinction between an agent's vote and its true preferences.) Conitzer and Sandholm [2005] studied elicitation algorithms for determining the winner under various voting rules (without any suspicion about how agents will vote), and gave lower bounds on the worst-case amount of information that agents must communicate.

3. ELICITING GENERAL PREFERENCES

As a basis for comparison, let us first analyze how difficult it is to elicit arbitrary (not single-peaked) preferences using comparison queries. We recall that our goal is to extract the agent's complete preferences, *i.e.* we want to know the agent's exact ranking of all m alternatives. This is exactly the same problem as that of sorting a set of m elements, when only binary comparisons between elements can be used to do the sorting. This is an extremely well-studied problem, and it is well-known that it can be solved using $O(m \log m)$ comparisons, for example using the MergeSort algorithm (which splits the set of elements into two halves, solves each half recursively, and then merges the solutions using a linear number of comparisons). It is also well-known that $\Omega(m \log m)$ comparisons are required (in the worst case). One way to see this is that there are $m!$ possible orders, so that an order encodes $\log(m!)$ bits of information—and $\log(m!)$ is $\Omega(m \log m)$. Hence, in general, *any* method for communicating an order (not just methods based on comparison queries) will require $\Omega(m \log m)$ bits (in the worst case).

Interestingly, for some common voting rules (including Borda, Copeland, and Ranked Pairs), it can be shown using techniques from communication complexity theory that even just determining whether a given alternative is the winner requires the communication of $\Omega(nm \log m)$ bits (in the worst case), where n is the number of agents [Conitzer and Sandholm, 2005]. That is, even if we do not try to elicit agents' complete preferences, (in the worst case) it is impossible to do more than a constant factor better than having each agent communicate all of its preferences! These lower bounds even hold for *nondeterministic* communication, but they do assume that preferences are unrestricted. By contrast, by assuming that preferences are single-peaked, we can elicit an agent's *complete* preferences using only $O(m)$ queries, as we will show in this paper. Of course, once we know the agents' complete preferences, we can execute any voting rule. This shows how useful it can be for elicitation to know that agents' preferences lie in a restricted class.

4. ELICITING WITH KNOWLEDGE OF ALTERNATIVES' POSITIONS

In this section, we focus on the setting where the elicitor knows the positions of the alternatives. Let $p : \{1, \ldots, m\} \rightarrow A$ denote the mapping from positions to alternatives, *i.e.* $p(1)$ is the leftmost alternative, $p(2)$ is the alternative immediately to the right of $p(1)$, ..., and $p(m)$ is the rightmost alternative. Our algorithms make calls to the function $\mathsf{Query}(a_1, a_2)$, which returns *true* if the agent whose preferences we are currently eliciting prefers a_1 to a_2, and *false* otherwise. (Since one agent's preferences are elicited at a time, we need not specify which agent is being queried.)

The first algorithm serves to find the agent's peak (most preferred alternative). The basic idea of this algorithm is to do a binary search for the peak. To do so, we need to be able to assess whether the peak is to the left or right of a given alternative a. We can discover this by asking whether the alternative immediately to the right of a is preferred to a: if it is, then the peak must be to the right of a, otherwise, the peak must be to the left of, or equal to, a.

```
FindPeakGivenPositions(p)

l ← 1
r ← m
while l < r {
  m1 ← ⌊(l + r)/2⌋
  m2 ← m1 + 1
  if Query(p(m1), p(m2))
    r ← m1
  else
    l ← m2
}
return l
```

Once we have found the peak, we can continue to construct the agent's ranking of the alternatives as follows. We know that the agent's second-ranked alternative must be either the alternative immediately to the left of the peak, or the one immediately to the right. A single query will settle which one is preferred. Without loss of generality, suppose the left alternative was preferred. Then, the third-ranked alternative must be either the alternative immediately to the left of the second-ranked alternative, or the alternative immediately to the right of the peak. Again, a single query will suffice—*etc.* Once we have determined the ranking of either the leftmost or the rightmost alternative, we can construct the remainder of the ranking without asking any more queries (by simply ranking the remaining alternatives according to proximity to the peak). The algorithm is formalized below. It uses the function $\mathsf{Append}(a_1, a_2)$, which makes a_1 the alternative that immediately succeeds a_2 in the current ranking (*i.e.* the current agent's preferences as far as we have constructed them). In the pseudocode, we will omit the (simple) details of maintaining such a ranking as a linked list. The algorithm returns the highest-ranked alternative; this is to be interpreted as including the linked-list structure, so that effectively the entire ranking is returned. c is always the alternative that is ranked last among the currently ranked alternatives.

```
FindRankingGivenPositions(p)

t ← FindPeakGivenPositions(p)
s ← p(t)
l ← t − 1
r ← t + 1
c ← s
while l ≥ 1 and r ≤ m {
  if Query(p(l), p(r)) {
    Append(p(l), c)
    c ← p(l)
    l ← l − 1
  } else {
    Append(p(r), c)
    c ← p(r)
    r ← r + 1
  }
}
while l ≥ 1 {
  Append(p(l), c)
  c ← p(l)
  l ← l − 1
}
while r ≤ m {
  Append(p(r), c)
  c ← p(r)
  r ← r + 1
}
return s
```

THEOREM 1. FindRankingGivenPositions *requires at most* $m - 2 + \lceil \log m \rceil$ *comparison queries.*

PROOF. FindPeakGivenPositions requires at most $\lceil \log m \rceil$ comparison queries. Every query after this allows us to add an additional alternative to the ranking, and for the last alternative we will not need a query, hence there can be at most $m - 2$ additional queries. □

Thus, the number of queries that the algorithm requires is linear in the number of alternatives. It is impossible to succeed using a sublinear number of queries, because an agent's single-peaked preferences can encode a linear number of bits, as follows. Suppose the alternatives' positions are as follows: $a_{m-1} < a_{m-3} < a_{m-5} < \ldots < a_4 < a_2 < a_1 < a_3 < a_5 < \ldots < a_{m-4} < a_{m-2} < a_m$. Then, any vote of the form $a_1 \succ \{a_2, a_3\} \succ \{a_4, a_5\} \succ \ldots \succ \{a_{m-1}, a_m\}$ (where the set notation indicates that there is no constraint on the preference between the alternatives in the set, that is, $\{a_i, a_{i+1}\}$ can be replaced either by $a_i \succ a_{i+1}$ or $a_{i+1} \succ a_i$) is single-peaked with respect to the alternatives' positions. The agent's preference between alternatives a_i and a_{i+1} (for even i) encodes a single bit, hence the agent's complete preferences encode $(m-1)/2$ bits. Since the answer to a comparison query can communicate only a single bit of information, it follows that a linear number of queries is in fact necessary.

5. ELICITING WITHOUT KNOWLEDGE OF ALTERNATIVES' POSITIONS

In this section, we study a more difficult question: how hard is it to elicit the agents' preferences when the alternatives' positions are not known? Certainly, it would be desirable to have elicitor software that does not require us to enter domain-specific information (namely, the positions of the alternatives) before elicitation begins, for two reasons: (1) this information may not be available to the entity running the election, and (2) entering this information may be perceived by agents as unduly influencing the process, and perhaps the outcome, of the election. Rather, the software should *learn* (relevant) information about the domain from the elicitation process itself.

It is clear that this learning will have to take place over the process of eliciting the preferences of multiple agents. Specifically, without any knowledge of the positions of the alternatives, the first agent's preferences could be *any* ranking of the alternatives, since any ranking is single-peaked with respect to some positions. Hence, eliciting the first agent's preferences will require $\Omega(m \log m)$ queries. Once the elicitor knows the first agent's preferences, though, some ways in which the alternatives may be positioned will be eliminated (but many will remain).

Can the elicitor learn the exact positions of the alternatives? The answer is no, for several reasons. First of all, we can invert the positions of the alternatives, making the leftmost alternative the rightmost, *etc.*, without affecting which preferences are single-peaked with respect to these positions. This is not a fundamental problem because the elicitor could choose either one of the positionings. More significantly, the agents' preferences may simply not give the elicitor enough information to determine the positions. For example, if all agents turn out to have the same preferences, the elicitor will never learn anything about the alternatives' positions beyond what was learned from the first agent. In this case, however, the elicitor could simply try to verify that the next agent whose preferences are to be elicited has the same preferences, which can be done using only a linear number of queries. More generally, one might imagine an intricate elicitation scheme which *either* requires few queries to elicit an agent's preferences, *or* learns something new and

useful from these preferences that will shorten the elicitation process for later agents. Then, one might imagine a complex accounting scheme, in the spirit of amortized analysis, showing that the total elicitation cost over many agents cannot be too large.

Fortunately, it turns out that we do not need anything so complex. In fact, knowing even *one* agent's (complete) preferences is enough to elicit any other agent's preferences using only a linear number of queries! (And a sublinear number will not suffice, since we already showed that a linear number is necessary even if we know the alternatives' positions.) To prove this, we will give an elicitation algorithm that takes as input one (the first) agent's preferences (*not* the positions of the alternatives), and elicits another agent's preferences using a linear number of queries.

First, we need a subroutine for finding the agent's peak. We cannot use the algorithm FindPeakGivenPositions from the previous section, since we do not know the positions. However, even the trivial algorithm that examines the alternatives one by one and maintains the most-preferred alternative so far requires only a linear number of queries, so we will simply use this algorithm.

```
FindPeak()

s ← a_1
for all a ∈ {a_2, ..., a_m}
  if Query(a, s)
    s ← a
return s
```

Once we have found the agent's peak, we next find the alternatives that lie between this peak, and the peak of the known vote (*i.e.* the peak of the agent whose preferences we know). The following lemma is the key tool for doing so.

LEMMA 1. *Consider votes v_1 and v_2 with peaks s_1 and s_2, respectively. Then, an alternative $a \notin \{s_1, s_2\}$ lies between the two peaks if and only if both $a \succ_{v_1} s_2$ and $a \succ_{v_2} s_1$.*

PROOF. If a lies between the two peaks, then for each i, a lies closer to s_i than s_{3-i} (the other vote's peak) lies to s_i. Hence $a \succ_{v_1} s_2$ and $a \succ_{v_2} s_1$. Conversely, $a \succ_{v_i} s_{3-i}$ implies that a lies on the same side of s_{3-i} as s_i (otherwise, v_i would have ranked s_{3-i} higher). But since this is true for both i, it implies that a must lie between the peaks. □

Thus, to find the alternatives between the peak of the known vote and the peak of the current agent, we simply ask the current agent, for each alternative that the known vote prefers to the peak of the current agent, whether it prefers this alternative to the known vote's peak. If the answer is positive, we add the alternative to the list of alternatives between the peaks.

The two votes must rank the alternatives between their peaks in the exact opposite order. Thus, at this point, we know the current agent's preferences over the alternatives that lie between its peak and the peak of the known vote (including the peaks themselves). The final and most complex step is to integrate the remaining alternatives into this ranking. (Some of these remaining alternatives may be ranked higher than some of the alternatives between the peaks.) The strategy will be to integrate these alternatives into the current ranking one by one, in the order in which the known vote ranks them, starting with the one that the known vote ranks highest. When integrating such an alternative, we first have the current agent compare it to the worst-ranked alternative already in the ranking. We note that the *known* vote must prefer the latter alternative, because this latter alternative is either the known vote's peak, or an alternative that we integrated earlier and that was hence preferred by the known vote. If the latter alternative is also preferred by the current agent, we add the new alternative to the bottom of the current ranking and move on to the next alternative. If not, then we learn something useful about the positions of the alternatives, namely that the new alternative lies on the *other side* of the current agent's peak from the alternative currently ranked last. The following lemma proves this.

LEMMA 2. *Consider votes v_1 and v_2 with peaks s_1 and s_2, respectively. Consider two alternatives $a_1, a_2 \neq s_2$ that do not lie between s_1 and s_2. Suppose $a_1 \succ_{v_1} a_2$ and $a_2 \succ_{v_2} a_1$. Then, a_1 and a_2 must lie on opposite sides of s_2.*

PROOF. If a_1 and a_2 lie on the same side of s_2—without loss of generality, the left side—then, because neither lies between s_1 and s_2, they must also both lie on the left side of s_1 (possibly, one of them is equal to s_1). But then, v_1 and v_2 cannot disagree on which of a_1 and a_2 is ranked higher. □

Knowing that the new alternative lies on the other side of the current agent's peak from the currently worst-ranked alternative will not help us to integrate the new alternative; in fact, our algorithm may still have to ask the agent to compare the new alternative to every alternative in the current ranking (other than the peak and the currently worst-ranked alternative). However, once we have integrated the new alternative, we know that *all alternatives that we integrate later must end up ranked below this alternative.* This is because of the following reason. Let us refer to the newly integrated alternative as c_1, and to the currently worst-ranked alternative as c_2. Because we have already taken care of the alternatives between the peaks of the current agent and the known vote, any later alternative that we integrate must lie on the same side of both peaks, on the same side as one of the two c_i. Because we are integrating alternatives in the order in which they are ranked by the known vote, the new (later) alternative must be further from the known vote's peak than that c_i. Hence, it must also be further from the current agent's peak than that c_i, so it must be ranked below c_1 by the current agent (since c_1 is ranked higher than c_2).

EXAMPLE 1. *Suppose the alternatives' positions are $e < c < b < f < a < d$. Also suppose that the known vote is $a \succ d \succ f \succ b \succ c \succ e$, and the preferences of the agent that we are currently eliciting are $c \succ e \succ b \succ f \succ a \succ d$. The algorithm will proceed as follows. First,* FindPeak *will identify c as the current agent's peak. Now, the alternatives that the known vote prefers to c are a (the known vote's peak), as well as d, f, b. For each of the last three alternatives, the algorithm queries the current agent whether it is preferred to a. The answer will be positive (only) for b and f, so we know that these two alternatives must lie between the peaks a and c on the line (and hence must be ranked oppositely by the known vote and the current agent). At this point, we*

know that the current agent must prefer $c \succ b \succ f \succ a$, and the algorithm must integrate d and e into this ranking. We set $c_1 = c$ and $c_2 = a$. The algorithm first integrates d since it is ranked higher than e in the known vote. The algorithm queries the agent with d and $c_2 = a$, and a is preferred. Now we know that the current agent must prefer $c \succ b \succ f \succ a \succ d$, and the algorithm sets $c_2 = d$. Finally, the algorithm must integrate e. The algorithm queries the agent with e and $c_2 = d$, and e is preferred. The algorithm then queries the agent with e and the successor of c_1, which is b. e is preferred, so the algorithm inserts e between $c_1 = c$ and b, and sets $c_1 = e$. At this point we know the entire ranking $c \succ e \succ b \succ f \succ a \succ d$.

We now present the algorithm formally. The algorithm again uses the function Append(a_1, a_2), which makes a_1 the alternative that immediately succeeds a_2 in the current ranking. It also uses the function InsertBetween(a_1, a_2, a_3), which inserts a_1 between a_2 and a_3 in the current ranking. The algorithm will (eventually) set $m(a)$ to *true* if a lies between the peaks of the current agent and the known vote v, or if a is the peak of v; otherwise, $m(a)$ is set to *false*. $v(i)$ returns the alternative that the known vote ranks ith (and hence $v^{-1}(a)$ returns the ranking of alternative a in the known vote, and $v^{-1}(a_1) < v^{-1}(a_2)$ means that v prefers a_1 to a_2). $n(a)$ returns the alternative immediately following a in the current ranking. Again, only the peak is returned, but this includes the linked-list structure and hence the entire ranking.

```
FindRankingGivenOtherVote(v)

s ← FindPeak()
for all a ∈ A
  m(a) ← false
for all a ∈ A − {s, v(1)}
  if v^{-1}(a) < v^{-1}(s)
    if Query(a, v(1))
      m(a) ← true
c_1 ← s
c_2 ← s
m(v(1)) ← true
for i = m to 1 step −1 {
  if m(v(i)) = true {
    Append(v(i), c_2)
    c_2 ← v(i)
  }
}
for i = 1 to m
  if not (m(v(i)) or v(i) = s)
    if Query(c_2, v(i)) {
      Append(v(i), c_2)
      c_2 ← v(i)
    } else {
      while Query(n(c_1), v(i))
        c_1 ← n(c_1)
      InsertBetween(v(i), c_1, n(c_1))
      c_1 ← v(i)
    }
return s
```

THEOREM 2. FindRankingGivenOtherVote *requires at most* $4m - 6$ *comparison queries.*

PROOF. FindPeak requires $m-1$ comparison queries. The next stage, discovering which alternatives lie between the current agent's peak and the known vote's peak, requires at most $m - 2$ queries. Finally, we must count the number of queries in the integration step. This is more complex, because integrating one alternative (which we may have to do up to $m - 2$ times) can require multiple queries. Certainly, the algorithm will ask the agent to compare the alternative currently being integrated to the current c_2. This contributes up to $m - 2$ queries in total. However, if the current alternative is preferred over c_2, we must ask more queries, comparing the current alternative to the alternative currently ranked immediately behind the current c_1 (perhaps multiple times). But every time that we ask such a query, c_1 changes to another alternative, and this can happen at most $m - 1$ times in total. □

In practice, the algorithm ends up requiring on average roughly $3m$ queries, as we will see in Section 6.

6. EXPERIMENTAL RESULTS

The following experiment compares FindRankingGivenPositions, FindRankingGivenOtherVote, and MergeSort. As discussed in Section 3, MergeSort is a standard sorting algorithm that uses only comparison queries, and can therefore be used to elicit an agent's preferences without any knowledge of the alternatives' positions or of other votes. Conversely, *any* algorithm that elicits general preferences using only comparison queries can be used to solve the sorting problem. So, effectively, if we want to compare to an algorithm that does not use the fact that preferences are single-peaked, then we cannot compare to anything other than a sorting algorithm. It is conceivable that other sorting algorithms perform slightly better than MergeSort on this problem, but they all require $\Omega(m \log m)$ comparisons.

In each run, first a random permutation of the m alternatives was drawn to represent the positions of the alternatives. Then, two random votes (rankings) that were single-peaked with respect to these positions were drawn. For each vote, this was done by randomly choosing a peak, then randomly choosing the second-highest ranked alternative from the two adjacent alternatives, *etc.* Each algorithm then elicited the second vote; FindRankingGivenPositions was given (costless) access to the positions, and FindRankingGivenOtherVote was given (costless) access to the first vote. (For each run, it was also verified that each algorithm produced the correct ranking.) Figure 1 shows the results. FindRankingGivenPositions outperforms FindRankingGivenOtherVote, which in turn clearly outperforms MergeSort.

One interesting observation is that FindRankingGivenOtherVote sometimes repeats a query that it has asked before. Thus, by simply storing the results of previous queries, the number of queries can be reduced. However, in general, keeping track of which queries have been asked imposes a significant computational burden, as there are $\binom{m}{2}$ possible comparison queries. Hence, in the experiment, the results of previous queries were not stored. FindRankingGivenPositions and MergeSort never repeat a query.

7. CONCLUSIONS

Voting is a general method for aggregating the preferences of multiple agents. Each agent ranks all the possible

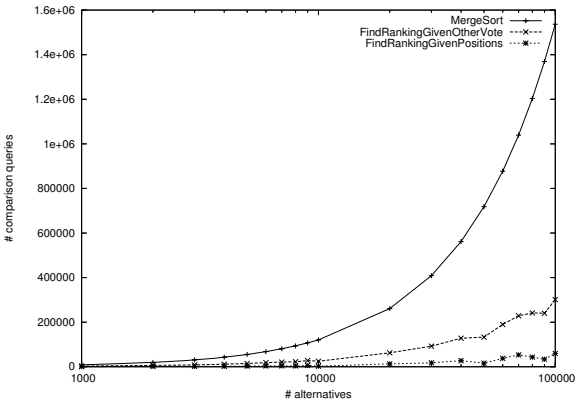

Figure 1: Experimental comparison of the two algorithms introduced in this paper, and MergeSort. Please note the logarithmic scale on the x-axis. Each data point is averaged over 5 runs.

alternatives, and based on this, an aggregate ranking of the alternatives (or at least a winning alternative) is produced. However, when there are many alternatives, it is impractical to simply ask agents to report their complete preferences. Rather, the agents' preferences, or at least the relevant parts thereof, need to be elicited. This is done by asking the agents a (hopefully small) number of simple queries about their preferences, such as comparison queries, which ask an agent to compare two of the alternatives. Prior work on preference elicitation in voting has focused on the case of unrestricted preferences. It has been shown that in this setting, it is sometimes necessary to ask each agent (almost) as many queries as would be required to determine an arbitrary ranking of the alternatives. By contrast, in this paper, we focused on single-peaked preferences. The agents' preferences are said to be single-peaked if there is some fixed order of the alternatives, the alternatives' *positions* (representing, for instance, which alternatives are more "left-wing" and which are more "right-wing"), such that each agent prefers alternatives that are closer to the agent's most preferred alternative to ones that are further away. We first showed that if an agent's preferences are single-peaked, and the alternatives' positions are known, then the agent's (complete) preferences can be elicited using a linear number of comparison queries. If the alternatives' positions are not known, then the first agent's preferences can be arbitrary and therefore cannot be elicited using only a linear number of queries. However, we showed that if we already know at least one other agent's preferences, then we can elicit the (next) agent's preferences using a linear number of queries (albeit a larger number of queries than the first algorithm). We also showed that using a sublinear number of queries will not suffice. Experimental results confirmed that these algorithms outperform algorithms that do not make use of the alternatives' positions or of previously elicited agents' preferences.

Future research includes studying elicitation in voting for other restricted classes of preferences. The class of single-peaked preferences (over single-dimensional domains) was a natural one to study first, due to both its practical relevance (real-world preferences often have this structure) and its useful theoretical properties (no Condorcet cycles and, as a result, the ability to aggregate preferences in a strategy-proof manner). Classes that are practically relevant but do not have these nice theoretical properties are still of interest, though. For example, one may consider settings where alternatives take positions in two-dimensional rather than single-dimensional space. It is well-known that in this generalization, Condorcet cycles can once again occur. Nevertheless, this does not imply that efficient elicitation algorithms do not exist for this setting. Nor does it imply that such elicitation algorithms would be useless, since it is still often necessary to vote over alternatives in such settings. However, if we use a voting rule that is not strategy-proof, then we must carefully evaluate the strategic effects of elicitation. Specifically, from the queries that agents are asked, they may be able to infer something about how other agents answered queries before them; this, in turn, may affect how they (strategically) choose to answer their own queries, since the rule is not strategy-proof. (This phenomenon is studied in more detail by Conitzer and Sandholm [2002].)

References

Kenneth Arrow. *Social choice and individual values*. New Haven: Cowles Foundation, 2nd edition, 1963. 1st edition 1951.

Duncan Black. On the rationale of group decision-making. *Journal of Political Economy*, 56(1):23–34, 1948.

Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5:649–667, 2004.

Ed H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 256–259, Tampa, FL, USA, 2001.

Vincent Conitzer and Tuomas Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 392–397, Edmonton, Canada, 2002.

Vincent Conitzer and Tuomas Sandholm. Communication complexity of common voting rules. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 78–87, Vancouver, Canada, 2005.

Allan Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

Sebastién Lahaie and David Parkes. Applying learning algorithms to preference elicitation. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, New York, NY, USA, 2004.

David Parkes. Iterative combinatorial auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 3. MIT Press, 2006.

Tuomas Sandholm and Craig Boutilier. Preference elicitation in combinatorial auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, chapter 10, pages 233–263. MIT Press, 2006.

Paolo Santi, Vincent Conitzer, and Tuomas Sandholm. Towards a characterization of polynomial preference elicitation with value queries in combinatorial auctions. In *Conference on Learning Theory (COLT)*, pages 1–16, Banff, Alberta, Canada, 2004.

Mark Satterthwaite. Strategy-proofness and Arrow's conditions: existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

Martin Zinkevich, Avrim Blum, and Tuomas Sandholm. On polynomial-time preference elicitation with value queries. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 176–185, San Diego, CA, USA, 2003.