Computational Social Choice: Spring 2009

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Plan for Today

Voting is the archetypical form of making a collective decision. As we have seen last week, there are a range of different voting rules, all either satisfying or violating various properties.

Today we will concentrate on some of the *computational* questions that arise in the context of voting. For instance:

- For a complex voting rule (think of the Dodgson rule), how do we actually *compute the winner* ($\leadsto$ algorithms)? And what is the computational complexity of doing so?
- The Gibbard-Satterthwaite Theorem tells us that *manipulation* is always possible. But how hard is it, computationally, to actually find a manipulating ballot?

Complexity of Manipulation in Voting

The motivation for studying the computational complexity of manipulation in voting is this:

- The Gibbard-Satterthwaite Theorem shows that manipulability is a universal problem in voting: there can always be a situation where a voter has an incentive not to vote sincerely, with all its repercussions ...
- But if it were computationally intractable to actually find out how to vote in order to manipulate successfully, then this may be deemed an acceptable risk.

Note: If it is hard to determine the winner for a voting rule, then it is likely to also be hard to manipulate that rule.

Complexity of Manipulation in Voting: Overview

The seminal paper by Bartholdi, Tovey and Trick (1989) starts by showing that manipulation is in fact *easy* for a range of commonly used voting rules, and then presents one system (a variant of the Copeland rule) for which manipulation is **NP**-complete.

- We first present a couple of these easiness results, namely for *plurality voting* and for the *Borda count*.
- We then present a result from a follow-up paper by Bartholdi and Orlin (1991): the manipulation of *single transferable vote* (STV) for electing a single winner is **NP**-complete.

J.J. Bartholdi III, C.A. Tovey, and M.A. Trick. The Computational Difficulty of Manipulating an Election. *Soc. Choice and Welfare*, 6(3):227–241, 1989.

J.J. Bartholdi III and J.B. Orlin. Single Transferable Vote Resists Strategic Voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

Manipulability as a Decision Problem

We can cast the problem of manipulability, for a particular voting rule f, as a decision problem:

MANIPULABILITY(f)
Instance: Set of ballots for all but one voter; candidate c.
Question: Is there a ballot for the final voter such that c wins?

We will be interested in the computational complexity of this problem in terms of the *number of candidates*.

Should MANIPULABILITY(f) be computationally intractable, then manipulation may be considered less of a worry for voting rule f.

Remark: We assume that the manipulator knows everybody else's ballot. This unrealistic assumption is intentional: if manipulation is intractable under such favourable circumstances, then that's great.

Manipulating the Plurality Rule

Recall the plurality rule:

- Each voter submits a ballot showing the name of one of the candidates. The candidate receiving the most votes wins.

The plurality rule is easy to manipulate (trivial):

- Simply vote for c, the candidate to be made winner by means of manipulation. If manipulation is possible at all, this will work. Otherwise not.

That is, we have $\textsc{Manipulability}(\mathit{plurality}) \in \mathbf{P}$.

<u>General:</u> $\textsc{Manipulability}(f) \in \mathbf{P}$ for any rule f with polynomial winner determination problem and polynomial number of ballots.

Borda Rule

Recall the Borda rule:

- Each voter submits a complete ranking of all the m candidates.
- For each voter that places a candidate first, that candidate receives $m-1$ points, for each voter that place her 2nd she receives $m-2$ points, and so forth.

  The Borda count is the sum of all the points.
- The candidate with the highest Borda count wins.

Remark: While the winner determination problem is polynomial, the number of valid ballots is superpolynomial for the Borda rule.

Manipulating the Borda Rule

The Borda rule is also easy to manipulate. Use a *greedy algorithm:*

- Place c (the candidate to be made winner through manipulation) at the top of your declared preference ordering.
- Then inductively proceed as follows: Check if any of the remaining candidates can be put next into the preference ordering without preventing c from winning. If yes, do so. If no, terminate and say that manipulation is impossible.

After having convinced ourselves that this algorithm is indeed correct, we also get $\textsc{Manipulability}(\mathit{Borda}) \in \mathbf{P}$.

Remark: Bartholdi *et al.* (1989) give a characterisation of a whole range of voting rules (including plurality and Borda), all of which are easy to manipulate.

J.J. Bartholdi III, C.A. Tovey, and M.A. Trick. The Computational Difficulty of Manipulating an Election. *Soc. Choice and Welfare*, 6(3):227–241, 1989.

Single Transferable Vote (STV)

Recall the STV system:

To select a *single winner*, it works as follows (voters submit ranked preferences for all candidates):

- If one of the candidates is the 1st choice for over 50% of the voters (*quota*), she wins.
- Otherwise, the candidate who is ranked 1st by the fewest voters (“plurality loser”) gets *eliminated* from the race.
- Votes for eliminated candidates get *transferred:* delete removed candidates from ballots and “shift” rankings (e.g., if your 1st choice got eliminated, then your 2nd choice becomes 1st).

Intractability of Manipulating STV

Theorem 1 (Bartholdi and Orlin, 1991) *Manipulation of STV for electing a single winner is* **NP***-complete.*

Proof sketch: Recall that proving **NP**-completeness requires proving **NP**-hardness and **NP**-membership. The latter is easy:

- Winner determination can be done in polynomial time (as the number of rounds is limited by the number of candidates).
- If someone *guesses* a preference ordering to be used for manipulation, we only need to run the polynomial winner determination algorithm to *check* whether it worked. ✓

As usual, the hard bit is to prove **NP**-hardness . . .

J.J. Bartholdi III and J.B. Orlin. Single Transferable Vote Resists Strategic Voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

Proving NP-hardness

The following decision problem is known to be NP-complete:

3-COVER
Instance: Sets $S_1, \ldots, S_m$ with $|S_i| = 3$; $S = \bigcup_{i=1}^{m} S_i$ with $|S| = n$.
Question: Is there an $I \subseteq \{1..m\}$ with $|I| = n/3$ and $\bigcup_{i \in I} S_i = S$?

The proof for **NP**-hardness of MANIPULABILITY(STV) works by reducing 3-COVER to the former: Given any instance of 3-COVER, we can construct an election which a manipulator can manipulate successfully iff he can solve the 3-COVER problem.

The proof itself is somewhat tedious. First define a long list of voter preferences (over $5m + n + 3$ candidates), carefully constructed such that one of two candidates will win. Then establish that, to make sure the one we don't want to win does not gain transferred votes, a whole list of other candidates need to stay in the game (blocking). This induces complex relationships between entries in the manipulator's ranking, which turn out to correspond to 3-COVER (see paper for details).

More on the Complexity of Voting

Much of the remainder of the lecture will be devoted to an overview of other recent results on the complexity of various problems arising in the context of voting.

(Almost) no proofs will be given, in most cases not even exact statements of technical results. The focus is on showing what kind of questions people have been asking.

Much of this material (winner determination, bribery, control) is covered in the survey paper by Faliszewski *et al.* (2006).

P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, and J. Rothe. *A Richer Understanding of the Complexity of Election Systems.* Technical Report TR-2006-903, Dept. of Computer Science, University of Rochester, 2006.

Winner Determination

For a given voting rule, what is the computational complexity of computing the winner for a given set of ballots?

In fact, we can distinguish several related problems:

- Compute the (or rather, a) winner of an election.
- Check whether a given candidate is a winner.
- Check whether one given candidate beats another given candidate according to some metric (e.g., the Borda count).
- Check whether a certain metric (score) for a given candidate is greater/less than some value K.

For a wide range of voting rules, all of these problems will be computationally easy.

Computing Dodgson Winners

The first paper studying the computational complexity of determining the winner of an election is another important paper by Bartholdi, Tovey and Trick (1989). Recall the *Dodgson rule:*

- A Dodgson winner is a candidate minimising the number of "switches" in the voters' linear preference orderings required to make that candidate a Condorcet winner.

Bartholdi *et al.* (1989) have shown that checking whether a candidate's Dodgson score exceeds K is **NP**-complete. Hemaspaandra *et al.* (1997) have refined this and shown that checking whether a candidate is a Dodgson winner it is *complete for parallel access to* **NP** (this requires some advanced complexity theory).

J.J. Bartholdi III, C.A. Tovey, and M.A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Soc. Choice Welf.*, 6(2):157–165, 1989.

E. Hemaspaandra, L.A. Hemaspaandra, and J. Rothe. Exact Analysis of Dodgson Elections. *Journal of the ACM*, 44(6):806–825, 1997.

Computing Banks Winners

If a voting rule allows for several winners (as most do, before tie-breaking), then it can happen that *checking* whether a particular candidate is one of the winners is *harder* than *finding some* winner. This is what happens for the *Banks rule:*

- Pairwise majority contests define a graph over candidates ("tournament"). A candidate is a Banks winner iff it is the top vertex in a maximal subgraph that is a linear order.

Woeginger (2003) has shown that checking if a given candidate is a Banks winner is **NP**-complete, while Hudry (2004) has shown that an arbitrary Banks winner can be computed in quadratic time.

G.J. Woeginger. Banks Winners in Tournaments are Difficult to Recognize. *Social Choice and Welfare*, 20(3):523–528, 2003.

O. Hudry. A Note on "Banks Winners in Tournaments are Difficult to Recognize" by G.J. Woeginger. *Social Choice and Welfare*, 23(1):113–114, 2004.

Bribery in Elections

When checking for *manipulability*, the name of the manipulator is part of the input. Similarly for a generalisation of the problem where we are checking for manipulability by a group of voters.

Bribery is the problem of finding $\leq K$ voters such that a suitable change of their ballots will make a given candidate c win.

Intuitively, bribery is harder (to check, not to perform!) than manipulation (because we also have to choose the manipulators).

See Faliszewski *et al.* (2006) to find out more ...

Controlling an Election

People have studied the computational complexity of a range of different ways of controlling an election:

- Adding or removing *candidates*.
- Adding or removing *voters*.
- In “electoral college”-style elections, redefining *districts* (if your party is likely to win with a huge majority in district A, it may be advantageous to merge part of it with district B . . .).

Control may be either *constructive* (to ensure a given candidate wins) or *destructive* (to ensure a given candidate does not win).

Again, see Faliszewski *et al.* (2006) to find out more . . .

Communication Complexity

We may also ask *how much information* voters need to transmit to allow us to compute the winner of an election.

Let n be the number of voters and m the number of candidates. We need $O(\log m)$ bits to communicate the identity of a candidate. Now upper bounds are easy to derive:

- Plurality: $O(n \log m)$ — each voter sends one candidate id
- Approval: $O(nm)$ — let each voter communicate a bit-string of length m to encode their approved subset of candidates
- Borda (any rule with linear orders as ballots): $O(n\,m \log m)$ — each voter sends m candidate id's in turn

Conitzer and Sandholm (2005) show that many of these bounds are tight, using tools from *communication complexity*.

V. Conitzer and T. Sandholm. *Communication Complexity of Common Voting Rules*. Proc. ACM Conference on Electronic Commerce 2005.

Upper Bound for STV

Recall that STV proceeds in rounds. In the extreme case, the full ranking of some voters may be required to compute the winner. Clearly, $O(n\, m \log m)$ is an upper bound.

Conitzer and Sandholm (2005) show that this is *not* a tight upper bound and give a better one (not tight either though): $O(n \log^2 m)$

- Protocol: in the first round, each voter names their favourite candidate; in subsequent rounds any voter whose favourite candidate has just been eliminated names their new favourite
- Insight: the number of voters who get their favourite candidate eliminated is small, namely $\leq \frac{n}{m-i+1}$ in round i
- $\sum_{i=1}^{m-1} \frac{n}{m-i+1} \;=\; n \cdot \sum_{i=2}^{m} \frac{1}{i} \;\leq\; n \ln m$ (harmonic number)
  So: at most $O(n \log m)$ transmissions of size $O(\log m)$ ✓

Other Computational Issues

- After some of the ballots have been counted, certain candidates may be *possible winners* or even *necessary winners*. How hard is it to check this? See e.g. Konczak and Lang (2005).
- ... and can we *compile* this partial vote information into a more compact form than just storing all the ballots (and still reason about it)? See Chevaleyre *et al.* (2008).
- We may *use computers* to systematically analyse (classical) questions in voting theory. For example, Trick (2006) analyses which voting rules are implementable by means of binary trees.

K. Konczak and J. Lang. *Voting Procedures with Incomplete Preferences.* Proc. Advances in Preference Handling 2005.

Y. Chevaleyre, J. Lang, N. Maudet, and G. Ravilly-Abadie. *Compiling the Votes of a Subelectorate.* Proc. COMSOC-2008.

M. Trick. *Small Binary Voting Trees.* Proc. COMSOC-2006.

Summary

This has been an introduction to computational issues in voting. We have concentrated on complexity results:

- Problems of which the complexity has been analysed: winner determination, manipulation, bribery, control
- *Winner determination* should be computationally easy.
- For the *manipulation*, *bribery* and *control* problems, intractability results are positive results.

  For manipulation, they suggest that the Gibbard-Satterthwaite Theorem may not matter that much in practice ... but beware that the quoted **NP**-hardness results are *worst-case* results; manipulation may well be easy *on average*.
- And: communication complexity, compilation complexity, ...

References

The following three papers are the main references for this lecture:

- J.J. Bartholdi III, C.A. Tovey, and M.A. Trick.
  The Computational Difficulty of Manipulating an Election.
  Social Choice and Welfare, 6(3):227–241, 1989.
- J.J. Bartholdi III and J.B. Orlin. Single Transferable Vote Resists Strategic Voting. *Social Choice and Welfare*, 8(4):341–354, 1991.
- P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, and J. Rothe. *A Richer Understanding of the Complexity of Election Systems*. Technical Report TR-2006-903, Dept. of Computer Science, University of Rochester, 2006.

Maybe rather surprisingly, you can get quite a lot out of all three of them even if you choose to skip all the technical bits!

What next?

Another issue of a computational nature in voting arises when the set of alternatives has got a *combinatorial* structure:

- Elect a committee of k members from n candidates.
- Vote in a referendum concerning n propositions.

We will address this problem two weeks from now, but first we will discuss how to model *preferences* in combinatorial domains.