Computing the Margin of Victory in IRV Elections

Thomas R. Magrino — *UC Berkeley*
Ronald L. Rivest — *MIT*
Emily Shen — *MIT*
David Wagner — *UC Berkeley*

Abstract

Efficient post-election audits select the number of machines or precincts to audit based in part on the margin of victory (the number of ballots that must be changed in order to change the outcome); a close election needs more auditing than a landslide victory. For a simple "first-past-the-post" election, the margin is easily computed based on the number of votes the first and second place candidates received. However, for instant runoff voting (IRV) elections, it is not immediately obvious how to compute the margin of victory. This paper presents algorithmic techniques for computing the margin of victory for IRV elections. We evaluate our method by attempting to compute the margin of victory for 25 IRV elections in the United States. The margin of victory computed can then be used to conduct post-election audits more effectively for IRV elections.

1 Introduction

In this paper, we study the problem of computing the margin of victory in instant runoff voting (IRV) elections. We define the margin of victory of an election to be the minimum number of ballots that, if they were different, would change the winner of the election.

The margin of victory of an election is a fundamental concept; it quantifies how close a race actually was. Also, the margin of victory suggests how much of a political mandate the winner can be thought of as receiving.

For a more traditional "first-past-the-post" election, the margin of victory is trivial to compute—it is half the difference in votes between the winner and the runner-up, rounded up if necessary. (For our purposes we assume that ties always break in a manner so as to produce a changed outcome; otherwise the "tie-free" margin may be one vote larger.)

For IRV elections, however, the margin of victory is much more complicated to compute. Since a ballot lists an order of preferences for candidates instead of a single choice of candidate, the number of ways a ballot can change is much greater. As we show later, the notion of a "runner-up" is not even quite the same as it is in a first-past-the-post election, since the candidate that loses the last runoff round is not necessarily the candidate who would win if the minimum number of changes were made to get another candidate to win.

The margin of victory is not only interesting in its own right, but it is also important for implementing efficient post-election audits. A post-election audit compares the electronic totals for a randomly-selected sample of precincts against the totals produced by hand-counting the paper ballots for those precincts. See Norden et al. [10] for an overview of post-election audits, or Joe Hall's excellent bibliography [9] for useful references. Post-election audits are useful in detecting errors or fraud; a "risk-limiting" post-election audit is further designed to limit the risk that the wrong candidate is named the winner. See Stark [13, 12, 14] for examples of risk-limiting audit methods.

Since hand-counting paper ballots is relatively expensive, an *efficient* post-election audit will audit relatively few precincts for a landslide election, but will audit more precincts when the election is close. The margin of victory measures how close the election is; it is thus a key input to the construction of an efficient post-election audit that provides a given level of statistical confidence.

In this paper, we develop algorithmic methods for computing the margin of victory in IRV elections. The natural algorithms for this problem run in time exponential in m, the number of candidates, because they involve exploring a space of size about $m!$ (the number of possible permutations of the m candidates). We achieve improved efficiency using several heuristics. Our core technical approach involves use of branch-and-bound methods to rapidly prune the space we need to explore. We explore the tree in a top-down fashion; at each internal node we check whether we can skip the entire subtree.

In particular, we show how to use integer linear programming to compute a lower bound on the margin of victory for an entire subtree, which in many cases allows us to prune exploration and skip that subtree. Our methods are heuristic in nature—the worst-case running time of our algorithm remains exponential in m—but we demonstrate that they are often effective in practice.

The primary result of this work is to (partially) remove one barrier to auditing of IRV elections. Some who are opposed to IRV claim that auditing IRV is intractable [4], due to the difficulty of precinct-based auditing of IRV elections. In contrast, some IRV supporters argue for single-ballot audits [5]. While single-ballot auditing of IRV elections may be feasible, there has not been a feasible way to compute the margin of victory in a IRV election, and therefore there does not currently seem to be an accepted method for performing single-ballot audits of IRV elections that provide a given level of statistical confidence.[1] Our work addresses this barrier to the auditing of IRV elections.

The contributions of this work are:

- We develop new algorithmic techniques to compute the margin of victory in IRV elections.
- We demonstrate experimentally that these algorithms are efficient in practice for many (but not all) real-world IRV elections.
- These results shed light into a broader policy debate surrounding IRV elections,[2] and suggest that IRV elections are not fundamentally incompatible with effective post-election audits.

2 Background: How IRV Elections Work

In this section we explain what IRV elections are and how they work.

In recent years, an increasing number of cities in the United States have adopted the IRV election format, which allows voters to rank the candidates in the order that the voter prefers the candidates [6]. In some places, such as San Francisco, the alternative name ranked choice voting (RCV) is used to refer to IRV. IRV elections have grown in popularity in recent years because they allow local officials to be elected with a majority vote without separate runoff elections.

[1] However, concurrent work by Sarwate et al. proposes an alternative approach to this problem [11].

[2] The selection of a voting system is *always* controversial. See `fairvote.org` for some favorable discussion and presentation of IRV, and `http://rangevoting.org/Irvtalk.html` for some discussion of some of IRV's rather surprising and undesirable properties. In this paper we focus on the auditability of IRV, and do not take a position on the larger policy question of whether IRV is a good voting system to be using in the first place.

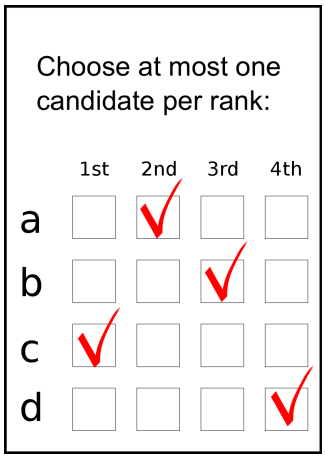

Figure 1: Example of a ballot for a voter who prefers c the most, followed by a, b, and d. We can think of this as the permutation (c,a,b,d).

In IRV, voters cast ballots that list candidates in order of preference. The ballot b contains a ranking $(c_1,c_2,\ldots,c_k)$ of a subset of the candidates, where c_1 is the voter's top choice, c_2 is the voter's second choice, and so on. For example, in an IRV contest with candidates a, b, c, and d, a voter who preferred c the most, followed by a, b, and d, would mark her ballot as shown in Figure 1. This ballot corresponds to the ranking (c,a,b,d). Voters do not need to rank all candidates; for instance, (b,c) is another possible ranking of candidates.

The winner is selected by simulating a series of runoff elections. In each runoff round, the candidate with the least number of votes is thereafter eliminated from contention, while the other candidates survive to the next runoff round. The winner is the last candidate remaining after all other candidates have been eliminated.

In each runoff round, votes are tallied by counting each ballot as a vote for its highest-ranked candidate who has not yet been eliminated. If a ballot has no eligible candidates listed, the ballot is *exhausted* and it does not count towards anyone, as if the ballot were cast blank. In each round, the candidate eliminated is the one with the lowest number of votes tallied in this fashion (with ties resolved according to local election law).

This procedure for determining the winner of an IRV contest is shown in pseudocode in Algorithm 1.

As an example, suppose we wanted to find the winner of an IRV contest between candidates a, b, c, and d with ballots cast as shown in Table 1. The series of runoff rounds and the number of votes that each candidate receives during each round is shown in Table 2.

Algorithm 1 Calculating the winner of an IRV election with candidates C and ballots B.

while $|C| > 1$ **do**
  Reset the tallies for each candidate $c \in C$ to 0.
  for each ballot $b = (c_1, c_2, \ldots, c_k) \in B$ **do**
    Add 1 to the tally of the highest-ranked candidate still in C.
  Eliminate the candidate with the smallest count from C.
return the remaining candidate in C.

Ranking of Candidates	Number of Ballots
(a,c,b,d)	40
(b,c,d,a)	21
(c,a,b,d)	10
(c,a,d)	10
(d,b,c,a)	5

Table 1: An example election between candidates a, b, c, and d. Each row shows the number of ballots cast with a specified ranking. Rankings are given in order of descending preference, so (a,b,c) means a was ranked first, b was ranked second, and c was ranked third.

3 Computing the Margin of Victory

The problem of finding the margin of victory for an IRV election is not a simple computation. Changing one ballot might have complex effects on the eventual winner; changing a single ballot might change which candidate is eliminated in an earlier round, which can have cascading effects on all subsequent rounds.[3]

Naively, one might be tempted to compute the margin of victory as the margin in the last round: i.e., one-half the difference between the number of votes allocated to the last two candidates in the final round. However, this method is not correct.

Consider the example shown in Tables 1 and 2. The last-round tie-free margin of victory is $(60-26)/2+1 = 18$, since by changing 18 ballots one can leave b with 44 votes in the last round and leave a with only 42 votes, causing b to win the election. However, the true margin of victory is much smaller than the last-round tie-free margin. By changing 4 ballots, we can change who is eliminated in the second round; we can cause b to be eliminated in the second round instead of c. When b is eliminated, all of the votes previously counted for b are now redistributed to c, and c wins the election. Therefore, the (tie-free) margin of victory can be much smaller than the (tie-free) last-round margin.

[3]See, for example, the example given by Warren Smith at `http://rangevoting.org/IRVamp.html`

	Number of Votes for			
Round	a	b	c	d
1	40	21	20	5
2	40	26	20	—
3	60	26	—	—

Table 2: An example of how the winner is determined, given the ballots from Table 1. If a candidate is eliminated before a runoff round, this fact is indicated by a dash. Candidate a wins the election.

Ranking of Candidates	Number of Ballots
(b,d)	71
(d)	10
(d,b)	5

Table 3: Example of reduced election profile for candidates b and d using the original profile in Table 1. All ballots are now treated as if they only listed b and d, omitting all other candidates they originally listed.

Similarly, we can see that the "apparent runner-up" (the last candidate to be eliminated) is not necessarily the candidate who missed winning by the smallest number of ballots.

Also, it is not hard to come up with examples that demonstrate that the margin of victory need not be equal to any of the individual round margins (i.e., between the candidate eliminated in that round and the candidate with the next higher total).[4]

Ties. One complexity of IRV is how ties are handled, in each round of the IRV algorithm. The method for handling ties is jurisdiction-specific and is not always fully specified, but it can potentially have a significant effect on the outcome of an election. In this paper, we conservatively assume that the adversary can control how ties are resolved: if in some round multiple candidates are tied for the lowest number of votes, we assume that the adversary can select which candidate will be eliminated. (If the adversary cannot in fact control the outcome of ties, then our methods might slightly underestimate the true margin of victory, but they will not overestimate it.)

3.1 Terminology

We start by defining a few terms we use in the description of our algorithms.

[4]For example, for the profile (a): 30, (b): 35, (c,a): 38, (d): 60, the margin of victory is 4 (changing votes from (c,a) to (a)), while the round margins are 3, 2, and 11.

- A **ballot signature** is the ranking of the candidates on the ballot. In other words, it is an ordered tuple of candidates, ordered from most preferred candidate to least preferred. In this paper, we allow the signature to be partial and not contain all of the candidates; unlisted candidates are understood to be preferred less than any listed candidate.

- The **election profile** is the list of ballots that are cast in the election. The election profile is a multiset of ballot signatures; it gives for each ballot signature the number of occurrences of that signature in the profile. Table 1 is an example of an election profile. In our problem, we are given an election profile (which corresponds to the publicly reported results), and we want to compute how many ballots would need to be modified to change the winner.

- The **elimination order** associated with an IRV election profile is a permutation of the candidates in the election. It lists candidates ordered by which round they are eliminated in, starting with the candidate that is eliminated earliest, and ending with the election winner (who isn't actually eliminated, but it is useful to list the winner here too). We can denote the elimination order as $(e_1, e_2, \ldots, e_m)$, where e_1 is the candidate that is eliminated earliest and e_m is the winner of the IRV contest. For the example in Table 1, the elimination order is (d, c, b, a).

- A ballot is **modified** when its signature is changed to a different signature.

  We assume that the election profile of the ballots as reported after the election is available and given to us. Thus, the original election profile is fixed and known; in calculating the margin of victory, we consider all potential modifications to this profile. (This is equivalent to considering profiles that might be the "true" election profile, from which our input election profile might have been derived by modifying ballots.)

- The **margin of victory** is the minimum number of ballot modifications that must occur in order for a different candidate to be declared the winner.

- Given an election profile P for a contest between candidates C, and given a subset of candidates $C' \subseteq C$, we call P' the **reduced election profile** for C' if it is the result of taking each ballot in P and removing any preferences involving candidates not in C'. For example, consider the election profile in Table 1: its reduced election profile for candidates b, d is shown in Table 3.

3.2 A Building Block

We start by considering a simpler problem:

Problem 1. *Given an election profile for an IRV election and a permutation of the candidates π, what is the minimum number of ballots one would have to change in order for the elimination order of the resulting election profile to be π?*

In other words, how many ballots must we modify if we want to force the elimination order to be π?

We use the solution to this problem as a subroutine in our full algorithm for the margin of victory problem.

Solution. Problem 1 can be solved using integer linear programming. In particular, one can construct an integer linear program whose objective function measures the number of changed ballots, and whose constraints enforce restrictions on what conditions must be satisfied in order for the new election profile's elimination order to be π. We now describe one way to construct an integer linear program to solve this problem.

For each ballot signature S, let n_S denote the number of ballots with signature S in the original election profile. Define $n = \sum_S n_S$, so that n counts the total number of ballots in the original election profile. The values of n_S and n are constants that are determined by the original election profile and thus are given to us.

We introduce a number of variables for the linear program. For each signature S in the new election profile, we introduce the variable p_S to represent the number of ballots that were modified so their new profile is S (and their original profile was something other than S). We also introduce the variable m_S to represent the number of ballots that were modified from their original profile S to some other profile. Finally, we introduce the variable y_S to represent the total number of ballots with signature S in the new election profile, after the modifications have been made.

The integer linear program then contains some basic inequalities that capture the form that modifications can take, and some special inequalities that require the elimination order of the modified profile to be π.

The basic inequalities are as follows. We add one instance of the following inequalities for each signature S:

$$\begin{aligned} n_S + p_S - m_S &= y_S \\ n \geq y_S &\geq 0 \\ n_S \geq m_S &\geq 0 \\ p_S &\geq 0 \end{aligned}$$

We also add the following inequality:

$$\sum_S p_S = \sum_S m_S$$

These inequalities enforce the following constraints:

- The number of ballots with new signature S is the number of ballots that originally had signature S, plus the number that were changed from something else to S, minus the number that changed from S to something else.
- The number of ballots that end up with signature S can not be more than the total number of ballots that were cast in the race.
- The number of ballots that are modified to have signature S must be nonnegative.
- We may not change more ballots of signature S than were originally reported.
- The number of ballots which originally had signature S and were then modified to have another signature must be nonnegative.
- The total number of ballots changed from something is equal to the total number of ballots changed to something (conservation of flow).

In addition to the basic inequalities, we also add several special inequalities to force the new election profile's elimination order to be $\pi = (e_1, e_2, \ldots, e_m)$. For $k \geq r$, let $\mathscr{S}_{k,r}$ be the set of ballot signatures that will be counted towards candidate e_k during round r, assuming that candidates are eliminated in the elimination order π. In other words, $\mathscr{S}_{k,r}$ is the set of all signatures such that, if we remove candidates $e_1, \ldots, e_{r-1}$ from the signature, e_k will then be ranked at the top. Put another way, $\mathscr{S}_{k,r}$ is the set of all signatures such that e_k is ranked at least as high as all of $e_r, e_{r+1}, \ldots, e_m$. Note that if candidates are eliminated following the elimination order π, then every ballot whose signature is in $\mathscr{S}_{k,r}$ will be counted towards candidate e_k in the rth round. With this definition, we are now prepared to describe the special inequalities. For each pair of distinct candidates e_i and e_j such that $i < j$, we generate the following inequality.

$$\sum_{S \in \mathscr{S}_{i,i}} y_S \leq \sum_{S \in \mathscr{S}_{j,i}} y_S$$

This inequality enforces the requirement that in the ith round, candidate j has at least as many votes as candidate i. Since in the ith round, the only candidates remaining are $e_i, e_{i+1}, \ldots, e_m$, taken collectively the inequalities for i imply that e_i must be the candidate eliminated in the ith round (assuming the previous rounds have followed π). Therefore, if all special inequalities hold, the elimination order will be exactly π.

Finally, the objective function is the number of ballots changed. We seek to minimize this objective function. In other words, the integer linear program's goal is

$$\text{minimize} \quad \sum_S p_S.$$

Ranking of Candidates	Number of Ballots
(a,b)	2
(b,a)	1

Table 4: A second example election profile for a contest between candidates a and b.

In summary, our initial solution to Problem 1 is to build the integer linear program described above and solve it using a standard ILP solver. Define a procedure `distanceTo`(π) that does exactly this; it returns the minimum number of ballots that must be changed in order to achieve an election profile whose elimination order will be π, starting from the original election profile. (The original election profile is exactly the set of reported ballots.) While the worst-case running time of an ILP solver is potentially exponential in the size of its input, in practice we have found that these ILPs can be solved efficiently.

Example. Consider the profile in Table 4. We start by adding the following constraint:

$$2 + p_{(a,b)} - m_{(a,b)} = y_{(a,b)}$$

This states that, after modifications are applied, the number of ballots that list a as the voter's first choice and b as her second choice is equal to 2 (the original number of ballots that had this order of preferences) plus the number of ballots that were changed to signature (a,b) minus the number of ballots that previously had (a,b) and were subsequently modified. Also, we would add the following constraints:

$$3 \geq y_{(a,b)} \geq 0$$
$$2 \geq m_{(a,b)} \geq 0$$
$$p_{(a,b)} \geq 0$$

These constraints state that, after modifications, the number of ballots with signature (a,b) must be non-negative and cannot exceed 3 (the total number of ballots in the election); the number of modified ballots that originally had signature (a,b) must be between 0 and 2, since we started with 2 ballots of this signature; and finally, the number of ballots modified to have signature (a,b) must be non-negative.

This captures the basic constraints for the signature (a,b). We would add similar constraints for the other signatures (b,a), (a), (b), and $()$. In addition, we would

add one more basic constraint to enforce that no ballots are added or removed from the race:

$$\sum_S p_S = \sum_S m_S$$

In other words, this equation is

$$p_{(a,b)} + p_{(b,a)} + p_{(a)} + p_{(b)} + p_{()} = m_{(a,b)} + m_{(b,a)} + m_{(a)} + m_{(b)} + m_{()}.$$

Suppose the desired elimination order is $\pi = (a,b)$, i.e., we wish to modify the ballots so a is eliminated first and b second. We obtain the special inequality

$$y_{(a)} + y_{(a,b)} \leq y_{(b)} + y_{(b,a)},$$

which simply states that, after modifications have been made, the total number of ballots that have b as the first choice must not be smaller than the total number of ballots that have a as the first choice.

Finally, we set the objective function to be the sum $\sum_S p_S$, which is the number of ballots changed. We use an integer linear programming solver to minimize the objective function, subject to the aforementioned constraints. In other words, our integer linear program (ILP) tries to achieve the new elimination order using the fewest changes to the original set of ballots.

3.3 An Unoptimized Margin Algorithm

Next, we introduce a basic way to compute the margin of victory for an IRV election. This approach is relatively simple but highly inefficient. The idea is to exhaustively enumerate all possible elimination orders that end with someone other than the currently declared winner. Then, for each such elimination order, we use our solution to Problem 1 (namely, the `distanceTo` procedure described in Section 3.2) as a subroutine to determine the smallest number of ballots that must be changed to achieve this elimination order. We dub this "the unoptimized algorithm" for computing the margin of victory.

The unoptimized algorithm examines each alternative elimination order that does not end with the reported winner and runs `distanceTo` on that order, finding the minimal number of changes needed to achieve that elimination order. It then returns the minimum value observed as the margin of victory. This corresponds to the elimination order that achieves a different winner with the fewest possible changes to the ballots. By definition, the value returned by the unoptimized algorithm is the margin of victory for an IRV election.

Examples. In the 2-candidate example election profile in Table 4 we can see the unoptimized algorithm does the same thing as what would be done in a "first-past-the-post" race. It considers the only other possible elimination order (a,b), in which a loses to b, and it finds the fewest number of modifications that would be needed for this elimination order to occur. In this case, the margin of victory is half the difference between the number of votes for a and b.

Algorithm 2 "Unoptimized algorithm" for finding the margin of victory for an IRV contest with candidates C, winner w, and election profile P.

```
bestSoFar ← ∞
for each permutation π of C that does not end with w
do
  dist ← distanceTo(π)
  if dist < bestSoFar then
    bestSoFar ← dist
return bestSoFar
```

For the election profile in Table 1, the unoptimized algorithm would iteratively call `distanceTo()` for every single permutation of the four candidates that does not designate a as the winner (since a was the originally reported winner). There are $3 \times 3! = 18$ such permutations of the four candidates, so Algorithm 2 invokes `distanceTo()` 18 times.

Efficiency analysis. The unoptimized algorithm is relatively inefficient, because it considers exponentially (in m) many elimination orders. In particular, there are $(m-1) \times (m-1)!$ permutations of the candidates that do not end with the originally reported winner, so Algorithm 2 performs $(m-1) \times (m-1)!$ iterations of its main loop. Thus, the number of calls to the `distanceTo()` procedure is $\Theta(m!)$, ensuring that the unoptimized algorithm will take time exponential in m.

Each iteration of the loop invokes an ILP solver. While in principle, this could itself take time exponential in the size of its input (since integer linear programming is NP-hard), in our experience the ILP solver's running time has typically not been the gating factor.

In the remainder of this paper, we focus on remedying the inefficiency of the unoptimized algorithm for contests with a large number of candidates, by trying to reduce the number of calls to `distanceTo()`.

3.4 Our Branch-and-Bound Algorithm

We now introduce an alternative approach that explores only a subset of the possible elimination orders. We use a branch-and-bound technique, motivated by the following lower bound.

Lemma 1. *Let $C = \{e_1, \ldots, e_m\}$ be a set of m candidates and $C' = C \setminus \{e_1\}$. Then,* `distanceTo`$((e_1, \ldots, e_m)) \geq$

`distanceTo`$((e_2,\ldots,e_m))$*, where the latter is calculated on the reduced election profile for* C'.

Proof. Let m_1 be the minimum number of ballots that need to be modified to achieve the elimination order $(e_1,\ldots,e_m)$ for profile P, and let m_2 be the minimum number of ballots that need to be modified to achieve the elimination order $(e_2,\ldots,e_m)$ for reduced profile P' (P restricted to candidates $\{e_2,\ldots,e_m\}$).

We wish to show that $m_2 \leq m_1$.

By definition of m_1, there is a way to change m_1 votes in P to get a new profile P_2 such that the elimination order for P_2 is $(e_1,\ldots,e_m)$. That is, P_2 satisfies the following properties:

- e_1 has the fewest first-choice votes.
- the reduced profile P_2' (P_2 restricted to candidates $\{e_2,\ldots,e_m\}$) has the elimination order $(e_2,\ldots,e_m)$.

Since we obtained P_2 from P by modifying m_1 ballots, we can obtain P_2' from P' by modifying at most m_1 ballots, by just restricting the ballot modifications between P and P_2 to the candidates $\{e_2,\ldots,e_m\}$. Therefore, m_1 is an upper bound on m_2. □

Corollary 1. `distanceTo`$((e_k,\ldots,e_m))$ *(calculated on the reduced election profile for candidate set* $\{e_k,\ldots,e_m\}$*) is a lower bound for* `distanceTo`$((e_1,\ldots,e_m))$.

This gives us a lower bound on `distanceTo`(π), for any elimination order π, which can be used in a branch-and-bound algorithm. The algorithm explores a search tree, whose leaves are full elimination orders for the candidates C, and whose internal nodes represent elimination orders for a reduced election profile (i.e., for a subset of the candidates). We use the lower bound in Lemma 1 to prune portions of the tree. When we can prove that an elimination order requires a larger number of modified ballots than the best one seen so far, we do not need to explore that elimination order.

In particular, we construct the search tree so that each internal node provides a lower bound on the values at all the leaves underneath it. We annotate each leaf with its corresponding `distanceTo()` value. For example, the leaf for elimination order $(e_1,\ldots,e_m)$ is annotated with `distanceTo`$((e_1,\ldots,e_m))$. Its parent corresponds to elimination order $(e_2,\ldots,e_m)$ of the reduced election profile for $\{e_2,\ldots,e_m\}$, and is annotated with `distanceTo`$((e_2,\ldots,e_m))$ (calculated for this reduced election profile).

The algorithm is designed to find the smallest leaf value. We keep track of the smallest leaf value seen so far, say v. Then, if we encounter an internal node whose value is v or greater, we can skip over the entire subtree rooted at that node, since none of its descendants can be smaller than the best seen so far.

Ranking of Candidates	Number of Ballots
(Alice, Clarence, Bob)	12
(Bob, Alice, Clarence)	6
(Clarence, Bob, Alice)	7

Table 5: A example election profile for a contest between candidates Alice, Bob, and Clarence.

As an example, consider the election profile in Table 1. The resulting search tree is shown in Figure 2. When visiting the node (b,d), we build the reduced election profile that includes only the candidates b and d and then compute `distanceTo`$((b,d))$ using this reduced profile.

In this example, we find `distanceTo`$((b,d)) = 28$, so we annotate the node (b,d) with the lower bound ≥ 28. Corollary 1 tells us that the leaves (a,c,b,d) and (c,a,b,d) will receive values ≥ 28. If we have previously explored a leaf with value less than or equal to 28, we do not need to explore the subtree rooted at (b,d).

Our branch-and-bound algorithm explores the search tree in "smallest-first" order. In particular, we prioritize nodes to explore based on the lower bound given by its parent, exploring nodes with smaller lower bounds earlier than nodes with greater lower bounds. We use a priority queue to store the "fringe" (the nodes that have not been explored yet), using each node's lower bound as its priority. This algorithm is designed to prune large parts of the search tree early by discovering a lower bound for them that is larger than the `distanceTo` value for some other leaf that has already been explored. If pruning is effective, many fewer calls to the ILP solver will be needed, potentially making the branch-and-bound algorithm considerably more efficient than the unoptimized algorithm. The pseudocode for our branch-and-bound algorithm is given in Algorithm 3.

We extend the basic branch-and-bound algorithm using heuristics to select which nodes to explore first when there is more than one with the same lower bound. These additional heuristics can be used as secondary priority keys to attach to each node as it is put into the priority queue. In our implementation of the algorithm, we chose to use 2 additional keys to prioritize nodes, which are described in Appendix A.

Example. Let us explore an example of how this would work for the election profile in Table 5, which shows a contest between the fictitious candidates Alice, Bob, and Clarence. We start the algorithm by placing the two reduced elimination orders (Bob) and (Clarence) on the priority queue (`fringe`) with priority zero.

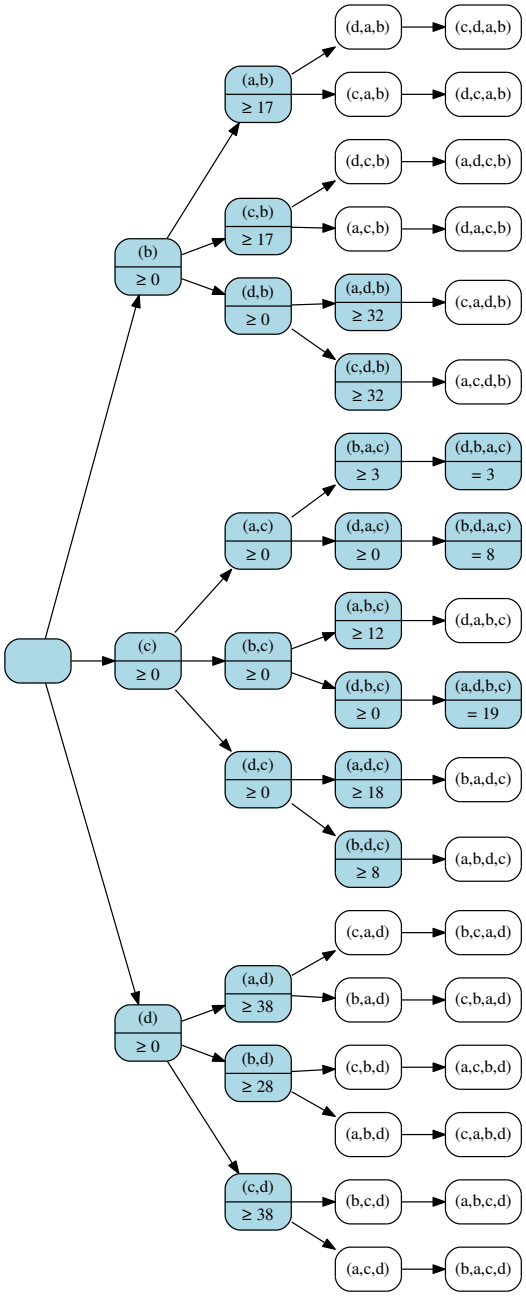

Figure 2: The search tree of reduced elimination orders for an election with candidates $C = \{a,b,c,d\}$, original winner a, and election profile given by Table 1. The nodes explored by our branch-and-bound algorithm are annotated with the `distanceTo()` value for that elimination order. Notation: (a,b) is annotated with ≥ 17, indicating that $\texttt{distanceTo}((a,b)) = 17$ (on the reduced election profile for $\{a,b\}$). This implies that (c,d,a,b) and (d,c,a,b) would receive a value at least 17, so they do not need to be explored. The final margin of victory computed was 3 ballots, corresponding to elimination order (d,b,a,c).

Algorithm 3 Our branch-and-bound algorithm for computing the margin of an IRV race with set of candidates C, winner w, and election profile P.

Let `fringe` be a priority queue.
for each candidate $c \in C \setminus \{w\}$ **do**
  Add (c) to `fringe` with priority 0.
while `fringe` is not empty **do**
  Pop the lowest priority elimination ordering in `fringe`. Suppose it is $\pi = (e_1, \ldots, e_k)$.
  if π is a complete elimination ordering **then**
    return $\texttt{distanceTo}(\pi)$.
  for each candidate $c \in C$ such that $c \notin \pi$ **do**
    Let $\texttt{d} \leftarrow \texttt{distanceTo}((c, e_1, \ldots, e_k))$ (computed on the reduced ballot profile for candidate set $\{c, e_1, \ldots, e_k\}$).
    Push $(c, e_1, \ldots, e_k)$ onto `fringe` with priority `d`.

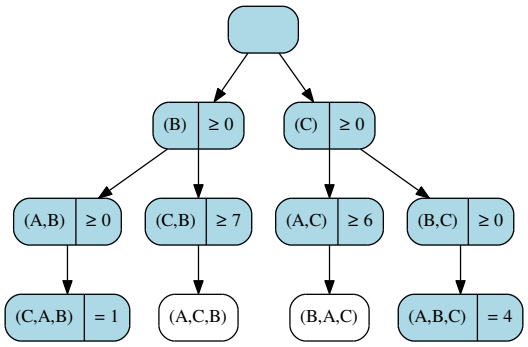

Figure 3: Tree corresponding to the example race between Alice, Bob and Clarence. We use the first initial of each candidate in the tree to save space.

We then move into the main loop and pop one of the two elimination orders, say (Clarence), from the fringe (either elimination order would work since they both have priority zero). We extend this elimination order by a single candidate in all possible ways to obtain the two orders (Alice, Clarence) and (Bob, Clarence). For each, we compute the reduced election profile and then invoke `distanceTo()`, receiving the values 6 and 0 respectively. We insert these two nodes into the fringe with these priorities.

In the next iteration, we pop the smallest node off the fringe, say (Bob). Again, we extend this by a single candidate to obtain the two extensions (Alice, Bob) and (Clarence, Bob), compute reduced election profiles, invoke `distanceTo()`, and insert them into the fringe with appropriate priorities, which are 0 and 7 respectively in this case. At this point, the fringe contains $\{(\text{Bob, Clarence}) \mapsto 0, (\text{Alice, Bob}) \mapsto 0, (\text{Alice, Clarence}) \mapsto 6, (\text{Clarence, Bob}) \mapsto 7\}$.

In the third iteration, we pop one of the lowest-priority elimination orders from the fringe, say (Bob, Clarence). The only possible extension is (Alice, Bob, Clarence). We invoke `distanceTo`((Alice, Bob, Clarence)), which returns 4, and insert it into the fringe with priority 4.

In the fourth iteration, we pop (Alice, Bob) and push (Clarence, Alice, Bob) with priority 1. In the fifth iteration, we pop (Clarence, Alice, Bob) and find that it is a complete ordering, so we are done and we know that the margin of victory for this race is 1. The corresponding search tree is displayed in Figure 3.

3.5 Reducing ILP Sizes

Next, we show another optimization that reduces the size of each ILP. It turns out that the basic approach in Section 3.2 introduces more variables than necessary. That approach introduces three variables per ballot signature. We show how to reduce the number of variables by introducing an equivalence relation on ballot signatures.

Fix an elimination order π. Consider a ballot signature S. Then all that matters about S is the n-tuple $f(S) = (c_1, \ldots, c_n)$, where c_i indicates which candidate the ballot S will be counted towards in the ith round of the IRV algorithm, when following elimination order π. If we have two ballot signatures S, S' where $f(S) = f(S')$, then we consider them equivalent: $S \sim S'$. As far as the IRV algorithm is considered, when we follow elimination order π and when $S \sim S'$, S and S' are interchangeable; they have the same effect. Consequently, all that matters is the total number of ballots in the equivalence class $[S]$ of ballot signatures equivalent to S.

For example, suppose we are examining elimination order (a,b,c). Then the ballot signature (b,a,c) is equivalent to the ballot signature (b,c): both ballot signatures count towards candidate b in the first two rounds, and then count towards candidate c in the third round. Therefore, a situation where we have 5 ballots with signature (b,a,c) and 3 ballots with signature (b,c) is equivalent to a situation where we have 8 ballots with signature (b,a,c) and none with signature (b,c). All that matters is the total number of ballots in this equivalence class.

This allows us to reduce the size of the ILP. Rather than introducing three variables p_S, m_S, y_S per ballot signature, we introduce three variables $p_{[S]}, m_{[S]}, y_{[S]}$ per equivalence class. We define $p_{[S]} = \sum_{S' \in [S]} p_{S'}$, and similarly for $m_{[S]}$ and $y_{[S]}$. Since the ILP of Section 3.2 can be expressed solely in terms of the $p_{[S]}, m_{[S]}, y_{[S]}$ variables, these are all we need.

This optimization significantly reduces the size of the ILPs we solve. For instance, in an election with m candidates, where each voter is allowed to rank all m candidates, the basic approach generates an ILP with $\Theta(m!)$ variables; this optimization reduces the number of variables to $\Theta(2^m)$ (there is one equivalence class per subset of the m candidates). In an election where each voter is only allowed to rank 3 candidates, the savings are smaller but still significant: the optimization reduces the number of variables from $3m(m-1)(m-2) + o(m^3)$ to $3\binom{m}{3} + o(m^3)$, an approximately 6-fold reduction. We have found that this optimization reduces the running time of our algorithm, by reducing the time it takes to solve each ILP.

4 Evaluation

In this section we evaluate the effectiveness of our branch-and-bound algorithm by using it to find the margin of victory for a variety of public IRV elections.

4.1 Dataset

We tested our branch-and-bound algorithm on 25 different elections.[5] Table 6 gives a detailed description of each. A few of the races were notably trivial in that they only had 2 candidates in the race, but we include the results since the elections were still run as IRV races (e.g., some voters placed both candidates on their ballot).

4.2 Methodology

We implemented our algorithm in C++ on an Intel Core i7 processor. We used the IBM CPLEX library with an academic research license as our integer linear programming solver. We tried several other ILP solvers; CPLEX

[5]The Aspen, CO data was obtained from http://irvfactcheckfactcheck.blogspot.com/2010/06/valid-ballot-what-does-it-mean-for-irv.html which is not affiliated with the Aspen City Clerk.

Location	Year	Office	m	Max cand. per ballot
Aspen, CO	2009	City Council	11	9
Aspen, CO	2009	Mayor	5	5
Berkeley, CA	2010	Auditor	2	3
Berkeley, CA	2010	District 1 City Council Rep.	5	3
Berkeley, CA	2010	District 4 City Council Rep.	5	3
Berkeley, CA	2010	District 7 City Council Rep.	4	3
Berkeley, CA	2010	District 8 City Council Rep.	4	3
Burlington, VT	2006	Mayor	6	6
Oakland, CA	2010	Auditor	3	3
Oakland, CA	2010	District 2 City Council Rep.	3	3
Oakland, CA	2010	District 2 School Board Dir.	2	3
Oakland, CA	2010	District 4 City Council Rep.	8	3
Oakland, CA	2010	District 4 School Board Dir.	3	3
Oakland, CA	2010	District 6 City Council Rep.	4	3
Oakland, CA	2010	District 6 School Board Dir.	2	3
Oakland, CA	2010	Mayor	11	3
Pierce County, WA	2008	City Council	4	3
Pierce County, WA	2008	County Assessor	7	3
Pierce County, WA	2008	County Executive	5	3
Pierce County, WA	2009	County Auditor	4	3
San Francisco, CA	2007	Mayor	18	3
San Leandro, CA	2010	District 1 City Council Rep.	4	3
San Leandro, CA	2010	District 3 City Council Rep.	2	3
San Leandro, CA	2010	District 5 City Council Rep.	3	3
San Leandro, CA	2010	Mayor	7	3

Table 6: The elections we tested our algorithm on. Here m denotes the number of candidates running in the contest, and the final column indicates the maximum number of candidates that a voter could rank on their ballot.

seemed to perform the best. We used the additional secondary keys described in Appendix A.

We used the local election board's rules for the number of candidates that could be listed on a single ballot in the race.[6] In many cases, voters were only permitted to rank up to 3 candidates. This reduces the number of variables created for each integer linear program, since there are fewer possible ballot signatures that could appear for the particular contest.

The main metric we used for evaluating the effectiveness of our algorithm was the total number of integer linear programs the algorithm had to solve before completing, regardless of the size of the integer linear program. Whenever feasible, we checked the computed answer against the solution returned by the "unoptimized algorithm" (Algorithm 2), in order to perform a "sanity check."

[6]We computed the margin of victory for all of these elections with respect to the IRV algorithm given in Algorithm 1, even though in some of the elections (e.g., Aspen, CO) the IRV rules used to run the actual election may have been slightly different. As a result, the margin of victory we have calculated for each race might differ from the actual margin of victory for that election's version of IRV.

4.3 Results

The running time of our algorithm is influenced most strongly by the number of candidates in the race and the number of candidates that each voter can rank. The number of candidates in the race directly determines the number of possible elimination orders, which strongly influences the number of ILPs that must be solved. The number of candidates that each voter can rank, along with the total number of candidates in the race, determines how many different ballot signatures are possible, which determines the number of variables in each ILP and thus has a strong influence on the time to solve each ILP.

When we were able to compute the margin, we saw reasonable performance: under 2 hours of computation time. Moreover, for large elections our branch-and-bound algorithm significantly reduced the number of integer linear programs that had to be solved, compared to the unoptimized algorithm. Table 7 summarizes the results of our performance evaluation.

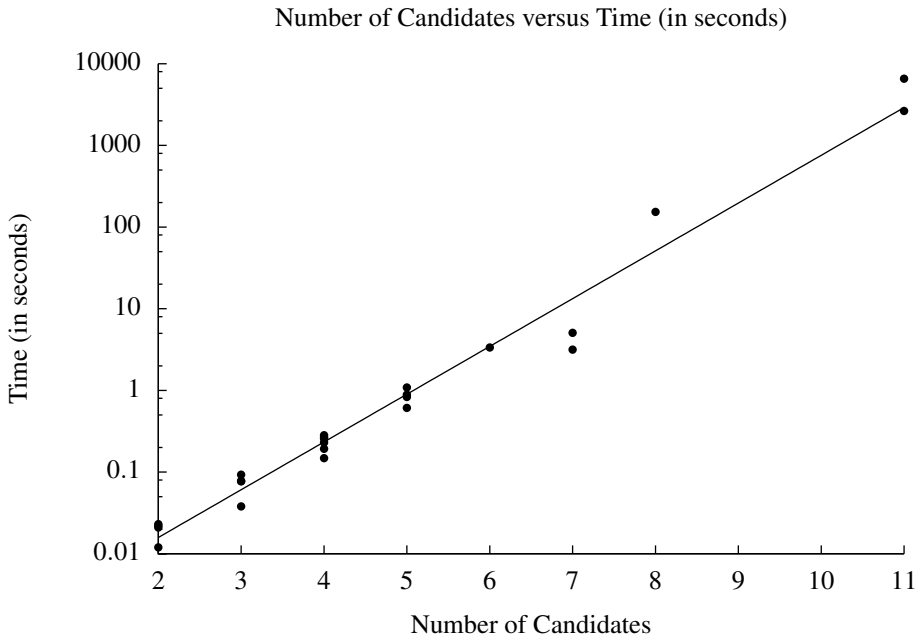

Figure 4: Graph of number m of candidates in a race versus the amount t of time in seconds to compute. The line shown is a best-fit line for t, namely, $t = e^{(am+b)}$ with $a = 1.3471$ and $b = -6.84145$.

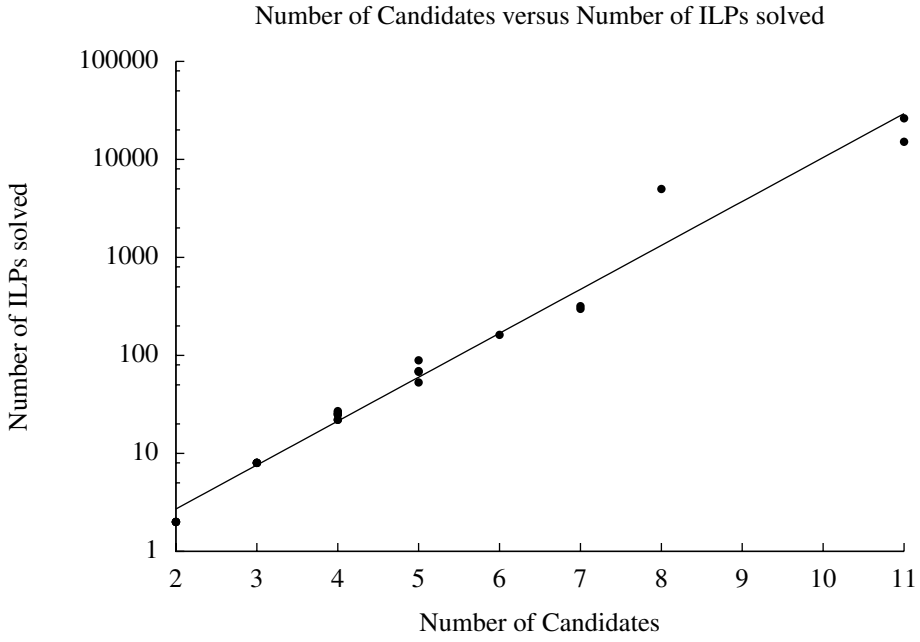

Figure 5: Graph of number m of candidates in a race versus the number K of ILPs we solved using our algorithm. The line shown is a best-fit line for K, namely, $K = e^{(am+b)}$ with $a = 1.03223$ and $b = -1.06821$.

m	n	Number of ILPs using Algorithm 2	Computing Time using Algorithm 2 (seconds)	Number of ILPs using Algorithm 3	Computing Time using Algorithm 3 (seconds)	Contest
2	45986	1	0.017	2	0.021	Berkeley 2010 Auditor
2	15243	1	0.017	2	0.012	Oakland 2010 D2 School Board
2	14040	1	0.018	2	0.023	Oakland 2010 D6 School Board
2	23494	1	0.017	2	0.022	San Leandro 2010 D3 City Council
3	122268	4	0.061	8	0.038	Oakland 2010 Auditor
3	15243	4	0.079	8	0.077	Oakland 2010 D2 City Council
3	23884	4	0.074	8	0.093	Oakland 2010 D4 School Board
3	23494	4	0.065	8	0.078	San Leandro 2010 D5 City Council
4	4862	18	0.397	25	0.267	Berkeley 2010 D7 City Council
4	5333	18	0.498	26	0.283	Berkeley 2010 D8 City Council
4	14040	18	0.599	27	0.148	Oakland 2010 D6 City Council
4	23494	18	0.297	25	0.257	San Leandro 2010 D1 City Council
4	43661	18	0.364	22	0.193	Pierce 2008 City Council
4	159987	18	0.310	22	0.230	Pierce 2009 County Auditor
5	312771	96	2.566	53	0.611	Pierce 2008 County Executive
5	2544	96	6.810	69	0.885	Aspen 2009 Mayor
5	6426	96	4.139	89	1.088	Berkeley 2010 D1 City Council
5	5708	96	4.052	68	0.831	Berkeley 2010 D4 City Council
6	9865	600	[Timeout]	162	3.358	Burlington 2006 Mayor
7	23494	4320	509.530	298	3.165	San Leandro 2010 Mayor
7	312771	4320	447.837	318	5.068	Pierce 2008 County Assessor
8	23884	35280	[Timeout]	4996	153.594	Oakland 2010 D4 City Council
11	122268	36288000	[Timeout]	26205	2646.488	Oakland 2010 Mayor
11	2544	36288000	[Timeout]	15119	6571.894	Aspen 2009 City Council
18	149465	6046686277632000	[Timeout]	[Timeout]	[Timeout]	San Francisco 2007 Mayor

Table 7: The running time of our algorithm (Algorithm 3, including the optimizations from Section 3.5) on various IRV elections. Algorithm 2 refers to the unoptimized algorithm. m denotes the number of candidates, and n denotes the number of ballots cast in the election.

Perhaps the biggest surprise was that, in all but two cases, the margin of victory exactly matched the last-round margin. The two exceptions were the 2008 Pierce County Assessor race and the 2009 Aspen City Council race. In both cases the margin of victory turned out to be less than one-third of the last-round margin. The margin of victory for each of the contests we were able to process is shown in Table 8.

4.4 Limitations of our Method

Our branch-and-bound algorithm was unable to compute the margin of victory for only one of the elections we tested, the 2007 San Francisco Mayoral election.

The 2007 San Francisco Mayoral contest had too many candidates for the algorithm to finish in a reasonable amount of time. Even with our pruning methods, the algorithm explores a very large number of tree nodes (elimination orders). In short, our branch-and-bound algorithm as it currently stands does not scale to IRV elections with large numbers of candidates.

We note that in some auditing approaches, it may be desirable to use the margin-of-victory computation multiple times, as it may be invoked each time a portion of the audit is completed. The efficiency of the margin-of-victory computation is particularly relevant in such cases.

5 Future Directions

In this section, we discuss some future directions for our work.

5.1 Improvements to our Algorithm

We suspect that further efficiency improvements to our algorithm may be possible. We list several possible di-

Contest	n	Margin	Last-round margin
Berkeley 2010 Auditor	45986	15356	15356
Oakland 2010 D2 School Board	15243	4830	4830
Oakland 2010 D6 School Board	14040	4826	4826
San Leandro 2010 D3 City Council	23494	8338	8338
Oakland 2010 Auditor	122268	17081	17081
Oakland 2010 D2 City Council	15243	2175	2175
Oakland 2010 D4 School Board	23884	3620	3620
San Leandro 2010 D5 City Council	23494	742	742
Berkeley 2010 D7 City Council	4862	364	364
Berkeley 2010 D8 City Council	5333	878	878
Oakland 2010 D6 City Council	14040	2603	2603
San Leandro 2010 D1 City Council	23494	3131	3131
Pierce 2008 City Council	43661	2007	2007
Pierce 2009 County Auditor	159987	8396	8396
Pierce 2008 County Executive	312771	2027	2027
Aspen 2009 Mayor	2544	89	89
Berkeley 2010 D1 City Council	6426	1174	1174
Berkeley 2010 D4 City Council	5708	517	517
Burlington 2006 Mayor	9865	388	388
San Leandro 2010 Mayor	23494	116	116
Pierce 2008 County Assessor	312771	1111	3650
Oakland 2010 D4 City Council	23884	2329	2329
Oakland 2010 Mayor	122268	1013	1013
Aspen 2009 City Council	2544	35	162

Table 8: The margin of victory in each election, computed using our algorithm. n denotes the number of ballots cast.

rections (as yet unimplemented and untested).

Weak candidates. In elections with many candidates, often there are many weak candidates, i.e., candidates who receive few votes (e.g., write-in candidates). These candidates cannot advance far, but our algorithm nonetheless examines many elimination orders where weak candidates survive into later rounds.

It is interesting to develop algorithmic optimizations that specifically target this situation. Fix a known upper bound u on the margin of victory. Fix an election profile P. For candidates a,b, let $n_*(a)$ denote the number of ballots that rank a higher than b (or that rank a anywhere and do not rank b at all), and let $n_1(b)$ denote the number of ballots that rank b as the first choice. Write $a \prec b$ if $n_*(a) + 2u < n_1(b)$. Note that $n_*(a)$ is an upper bound on the number of ballots that might count towards a in any round of the IRV algorithm executed on profile P, and $n_1(b)$ is a lower bound on the number of ballots that might count for b. If these differ by more than $2u$, then no modification of u or fewer ballots can cause a to be eliminated later than b. In other words, if $a \prec b$ and $\texttt{distanceTo}(\pi) \leq u$, then a must occur before b in π, i.e., $\pi = (\ldots, a, \ldots, b, \ldots)$. Consequently, we can use the partial order $\prec$ to narrow our search.

This suggests an algorithm. Initially, set u to the last-round margin and compute the relation $\prec$ by examining all $m(m-1)$ pairs of candidates. Say that π is inconsistent with $\prec$ if there exist a pair of candidates $a \prec b$ such that either (i) b appears before a in π, or (ii) π is a partial elimination order that mentions a but not b. Now, search the tree of elimination orders as in the branch-and-bound algorithm, but skipping over nodes (and their subtrees) that are inconsistent with $\prec$. At any point where we find a complete elimination order whose `distanceTo` value is smaller than u, we can optionally update u and $\prec$. It seems plausible that, when there are many weak candidates and only a few strong candidates, this optimization might significantly improve the performance of our algorithm—however, we have not tested it.

Further bounds. In Section 3.4, we showed how to lower-bound $\texttt{distanceTo}((e_1, \ldots, e_m))$ by examining a suffix $(e_k, \ldots, e_m)$ of the elimination order. We note that one can also obtain lower bounds from a prefix as well.

More formally, if $\{c_1, \ldots, c_k\}, B$ are two candidate-sets whose disjoint union is C, let $\langle c_1, \ldots, c_k, B \rangle$ denote the set of all elimination orders of the form

$(c_1,\ldots,c_k,b_1,\ldots,b_{m-k})$ such that $b_1,\ldots,b_{m-k}$ forms a permutation of B. Given $\langle c_1,\ldots,c_k,B\rangle$, we can form an ILP problem as described in Section 3.2, but omitting the constraints corresponding to the last $m-k$ rounds of the IRV election. Let $\texttt{distanceTo}(\langle c_1,\ldots,c_k,B\rangle)$ denote the result of solving this ILP. We note that $\texttt{distanceTo}(\langle c_1,\ldots,c_k,B\rangle) \leq \texttt{distanceTo}((c_1,\ldots,c_k,b_1,\ldots,b_{m-k}))$ for all permutations of B. Consequently, we obtain a lower bound analogous to Corollary 1, but for the prefix rather than the suffix.

More generally, we can also combine these two ideas, to obtain a lower bound on $\texttt{distanceTo}(\pi)$ corresponding to each consecutive sequence of candidates from π. If $A,\{c_1,\ldots,c_k\},B$ are three candidate-sets whose disjoint union is C, let $\langle A,c_1,\ldots,c_k,B\rangle$ denote the set of all elimination orders of the form $(a_1,\ldots,a_{|A|},c_1,\ldots,c_k,b_1,\ldots,b_{|B|})$ where the a's form a permutation of A and the b's a permutation of B. We can compute the reduced election profile for candidates $\{c_1,\ldots,c_k\}\cup B$, build an ILP but omitting constraints corresponding to the last $|B|$ rounds, and then solve the IRV to obtain $\texttt{distanceTo}(\langle A,c_1,\ldots,c_k,B\rangle)$. The key fact is that $\texttt{distanceTo}(\langle A,c_1,\ldots,c_k,B\rangle) \leq \texttt{distanceTo}((a_1,\ldots,a_{|A|},c_1,\ldots,c_k,b_1,\ldots,b_{|B|}))$ always holds.

Thus, for each elimination order π, we obtain many lower bounds on $\texttt{distanceTo}(\pi)$. It seems conceivable that there may be a way to build a more efficient branch-and-bound algorithm that uses all of these bounds simultaneously to prune the search tree, rather than just Corollary 1. However, we have not explored this direction.

5.2 Computational Complexity

While we have, as a practical matter, made progress on the problem of computing the margin of an IRV election, it is still open whether this problem is actually solvable in time polynomial in the number m of candidates and the number n of votes cast.

There has been a substantial body of research, beginning with a series of works by Bartholdi, Orlin, Tovey, and Trick [2, 1, 3], studying the worst-case computational complexity of manipulation (strategic voting) problems for various voting rules. See [7] and [8] for overviews of this line of research. In particular, Bartholdi and Orlin [1] showed that the problem of manipulating of IRV is NP-complete. Given this earlier work and the similarity between strategic voting problems and the margin problem, it is natural to conjecture that computing the margin of victory of an IRV election is also NP-complete. However, we have not yet proven this conjecture.

6 Conclusions

The margin of victory is a fundamental notion for post-election audits, because many post-election auditing schemes require knowledge of the margin of victory in order to design an efficient sampling strategy that is guaranteed with high probability to catch (and, for a risk-limiting audit, correct) errors sufficient to have changed the election outcome. Also, the margin of victory has public and political relevance, because it quantifies how close the election was.

In this paper, we presented a relatively efficient method for computing the margin of many IRV elections. Our algorithm is able to compute the margin of victory for all but one of the real-world IRV elections we tested.

In one case we were unable to compute the margin of victory. This failure could be attributed to an unusually large number of candidates in the race (which usually does not occur in smaller cities for local elections). In most cases, however, we show that our algorithm works relatively efficiently in practice. Our results show that, in most cases, the margin of victory could be determined within a few hours after the election is tabulated.

Acknowledgements

This work was partially supported by National Science Foundation grant CNS-0524745. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] BARTHOLDI, J. J., AND ORLIN, J. B. Single transferable vote resists strategic voting. *Social Choice and Welfare 8*, 4 (1991), 341–354.

[2] BARTHOLDI, J. J., TOVEY, C. A., AND TRICK, M. A. The computational difficulty of manipulating an election. *Social Choice and Welfare 6* (1989), 227–241.

[3] BARTHOLDI, J. J., TOVEY, C. A., AND TRICK, M. A. How hard is it to control an election. *Mathematical and Computer Modelling 16* (1992), 27–40.

[4] DOPP, K. Realities mar instant runoff voting - 18 flaws and 4 benefits. `http://electionmathematics.org/ucvAnalysis/US/RCV-IRV/InstantRunoffVotingFlaws.pdf`, Feb. 2009.

[5] Ranked voting and election integrity. `http://archive.fairvote.org/?page=2438`, Dec. 2009.

[6] Where instant runoff is used. `http://www.fairvote.org/where-instant-runoff-is-used`, 2010.

[7] FALISZEWSKI, P., HEMASPAANDRA, E., AND HEMASPAANDRA, L. A. Using complexity to protect elections. *CACM 53*, 11 (Nov 2010), 74–82.

[8] FALISZEWSKI, P., AND PROCACCIA, A. D. AI's war on manipulation: Are we winning? *AI Magazine 31*, 4 (Dec 2010), 53–64.

[9] HALL, J. L. Post-election manual auditing of paper records: Bibliography. http://www.josephhall.org/papers/auditing_biblio.pdf, 2007.

[10] NORDEN, L., BURSTEIN, A., HALL, J. L., AND CHEN, M. Post-election audits: Restoring trust in elections. Brennan Center for Justice at The New York University School of Law and The Samuelson Law, Technology and Public Policy Clinic at the University of California, Berkeley School of Law (Boalt Hall), Aug. 2007.

[11] SARWATE, A., CHECKOWAY, S., AND SHACHAM, H. Risk-limiting audits for nonplurality elections. Tech. Rep. CS2011-0967, UC San Diego, June 2011. https://cs.ucsd.edu/~scheckow/papers/nonplurality2011.html.

[12] STARK, P. B. Efficient post-election audits of multiple contests: 2009 california tests. Refereed paper presented at 2009 Conference on Empirical Legal Studies.

[13] STARK, P. B. Super-simple simultaneous single-ballot risk-limiting audits. In *2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE'10)*. http://www.usenix.org/events/evtwote10/tech/full_papers/Stark.pdf.

[14] STARK, P. B. Risk-limiting post-election audits: P-values from common probability inequalities. *IEEE Transactions on Information Forensics and Security 4* (2009), 1005–1014.

A Additional Priority Values

As mentioned in Sections 3.4 and 4.2, we used the `distanceTo` value as the primary priority for each node, and we used secondary and tertiary priority values to break ties.

We used the multiplicative inverse of the number of candidates included in the elimination order as the secondary priority. The goal is to quickly find a possible full solution early on in the algorithm (i.e., to quickly reach a leaf). This key causes our algorithm to simulate a depth-first search among all possible nodes of the same primary priority, if there are many nodes with the same primary priority.

Our tertiary priority for a partial elimination order of length k was the number of candidates in this order which were not found in the last k rounds of the reported elimination order. For example, the node (d,b) would receive a tertiary priority value of 1, since the last two rounds of the reported elimination involved (b,a). This choice of tertiary priority was based on a notion that the margin of victory might be more likely occur in an elimination order that is similar to the original, reported elimination ordering.

However, these secondary and tertiary priorities did not yield a significant improvement for the algorithm's performance, compared to using only the primary priority.