Democracy Developers
A Guide to Risk Limiting Audits with RAIRE
Part 2: Generating Assertions with RAIRE

Michelle Blom, Peter Stuckey and Vanessa Teague

Date: August 30, 2023
Version 1.0 DRAFT

Contents

Executive Summary

This document is intended to help election administrators, candidates and interested members of the public understand how to conduct Risk Limiting Audits (RLAs) for Instant Runoff Voting (IRV) using RAIRE.

Risk limiting audits (RLAs) are a post-election activity, performed to provide a certain level of confidence in the correctness of the reported outcome, or to correct a wrong outcome by manual recount. These audits involve randomly sampling paper ballots that have been cast by voters. After collecting and analysing such a sample, statistical computations are performed to ascertain a current level of *risk*. A Risk Limiting Audit guarantees a *Risk Limit*: the maximum probability that it will mistakenly confirm a reported outcome that was in fact wrong. The audit proceeds by sampling ballots until this risk falls below an acceptable level, or election administrators determine that a full manual count is required.

Part 1 of this guide gives a general overview of the audit process incorporating RAIRE, reviews instant runoff voting (IRV), explains how we visualize and analyze IRV elections, defines the main concept in RAIRE audits–*assertions*–and explains how to conduct a complete RLA for IRV elections using assertions generated by RAIRE.

This document presents Part 2 of this guide, providing the details on how RAIRE generates assertions (Chapter 1), and optimizations used to make this process more efficient (Chapter 2). Appendix 2 presents additional details on the scoring of assertions during an IRV RLA, and their verification.

Chapter 1

Using RAIRE to generate assertions

Sometimes the assertions required to verify that the announced winner in an IRV election is correct can be picked by a human, but sometimes an IRV election is complex enough to need automated analysis to derive a set of assertions that are true and imply that the announced winner won.

RAIRE's Objectives

RAIRE has three goals:

1. It wants to find a set of assertions that, if true, rule out all outcomes in which someone other than the announced winner wins. It wants to find a *valid* audit.

2. It wants to minimize the number of ballots that auditors need to collect and examine in order to verify all of these assertions – it wants to find a *least-cost valid* audit.

3. It wants to generate these assertions in a computationally efficient way.

To explain how RAIRE achieves these objectives, let's consider an example election between Alice, Bob, Chuan, and Diego. In Part 1, our examples involved only a handful of ballots. We now consider a larger election with thousands of ballots. Chuan is the winner of this IRV election, with Bob, Diego, and Alice eliminated in that order (i.e., Alice is the runner-up).[1] We will start by outlining how the unoptimized version of RAIRE generates assertions for this election. In Chapter 2, we will explain how a key optimization allows RAIRE to find an appropriate set of assertions more efficiently.

[1]We'll use the term "runner-up" for the candidate who is second-last in the elimination order, though of course that candidate might not be the loser who is closest to winning.

Example 1. *Suppose there are 4 candidates: Alice, Bob, Chuan and Diego. The votes are as follows:*

Preferences	***Count***
(C, B, A)	*5000*
(B, C, D)	*1000*
(D, A)	*1500*
(A, D)	*4000*
(D)	*2000*

Alice, Bob, Chuan and Diego initially have 4000, 1000, 5000, and 3500 votes, respectively. Bob has the smallest tally and is eliminated first, with his 1000 ballots transferred to Chuan. Alice, Chuan and Diego now have 4000, 6000, and 3500 votes, respectively. Diego is eliminated, giving 1500 votes to Alice. Chuan, still on 6000 votes, has more votes than Alice, on 5500, and is declared the winner.

1.1 Exploring alternative outcomes

To determine what assertions we need in order to audit this IRV election, RAIRE visualizes the space of alternate election outcomes–in which a candidate other than the reported winner wins–as a collection of trees, as described in Part 1. Then, RAIRE generates assertions that rule out all outcomes in these 'alternate outcomes trees' by attacking each of their branches at their *weakest point*. To do so, RAIRE uses the election's CVRs to assess how each branch can be ruled out with an assertion that is likely to require a small sample to audit. For elections with a small number of candidates, we can easily create a complete set of elimination trees, visualizing all possible elimination orders ending with an alternate winner. For large elections, it would be computationally intractable to build and analyze full elimination trees. RAIRE applies a method that allows us to partially construct these trees, while at the same time deriving assertions that demonstrate that none of the alternative outcomes is the true one.

RAIRE starts by considering only the first level of the full set of alternate outcomes, visualized in Figure 1.1. Recall that the reported winner is Chuan. We do not consider any outcome that ends with Chuan winning. Our goal is to rule out all outcomes in which Alice, Bob, or Diego win.

First Level of Alternate Outcomes Trees

Alice
Bob
Diego
1
2
3

Figure 1.1: First level of nodes in our alternate outcome tree set for the election in Example 1.

We can attack nodes at the first level of our tree set with NEB assertions. If we can show that there is a candidate that *cannot be eliminated before* Alice, for example, this rules out any outcome that has

Alice as the winner. For each node at this first level, we find the easiest to audit assertion, requiring the smallest sample size, that: holds on the basis of the CVRs; and that attacks the associated outcome. Each assertion is assigned a numeric 'difficulty', representing how 'easy' it will be to audit. We expect that assertions with a higher difficulty will require a larger ballot sample to audit. For each node, we take the easiest to audit assertion that attacks it, and assign it to the node along with its difficulty. If we cannot find an assertion to attack a node, we assign the node a difficulty of infinity (∞).

RAIRE can be used with any supplied function to compute assertion difficulty–or 'ease of auditing'. For ballot level comparison audits, one option is to use *1 divided by the diluted margin* of the assertion to estimate its difficulty. For ballot polling audits, one option is to use the formula *1 divided by the square of the diluted margin* of the assertion to estimate its difficulty.

Estimating the difficulty of auditing an assertion

Given a set of assertions that attacks an outcome, the 'easiest to audit' assertion is the one we expect would require the smallest sample of ballots to audit. Given two assertions, we assume that the assertion with the larger diluted margin will be the easier one to audit. For the purposes of our running example, we will define the diluted margin of an assertion with winner w and loser l as the difference in tallies of the winner and loser divided by the total number of ballots cast. Recall that when we talk about the winner and loser of an assertion, we are not referring to the ultimate winners and losers of the election – we are just referring to the candidates being compared by the assertion itself. Also recall that what ballots sit in the respective tallies of this winner and loser when performing this comparison depends on the rules of the assertion. In our example, the total number of cast ballots is 13500. Let's also assume that we are undertaking a ballot level comparison audit, and use *1 divided by the diluted margin* to estimate the auditing difficulty of a given assertion.

RAIRE first considers node 1 (Alice) in Figure 1.1, and looks for NEB assertions to show that there is a candidate that could never be eliminated while Alice is continuing, as shown in Table 1.1a.

Table 1.1a looks at all NEB assertions that could possibly be formed with Alice as the loser. Recall that NEB assertions compare the smallest possible tally the winner (in the context of the assertion) could have against the largest possible tally that the loser could ever attain. In the first entry of Table 1.1a, we have chosen Bob as the winner. Bob's minimum tally is his first preference count of 1000 votes. Alice's maximum tally counts all ballots where Alice is ranked higher than Bob, or where Alice is ranked while Bob has not been preferenced on the ballot. Alice's maximum tally in this context is 5500 ballots. This includes the 4000 (A, D) ballots and the 1500 (D, A) ballots. All other ballots either rank Bob higher than Alice, or do not preference Alice at all. It is clear that we can not form any NEB assertions to rule out Alice as a winner, as Alice's maximum tally in all three possibilities is greater than the minimum tallies of Bob, Chuan, and Diego. So, at this point, RAIRE will assign node 1 (Alice) a difficulty of infinity.

Assertion	Winner	Loser	Minimum Winner Tally	Maximum Loser Tally	Difficulty
Bob NEB Alice	Bob	Alice	1000	5500	–
Chuan NEB Alice	Chuan	Alice	5000	5500	–
Diego NEB Alice	Diego	Alice	3500	9000	–

(a) NEB assertions to show that Alice cannot be the winner.

Assertion	Winner	Loser	Minimum Winner Tally	Maximum Loser Tally	Difficulty
Alice NEB Bob	Alice	Bob	4000	6000	–
Chuan NEB Bob	Chuan	Bob	5000	1000	3.375
Diego NEB Bob	Diego	Bob	3500	6000	–

(b) NEB assertions to show that Bob cannot be the winner.

Assertion	Winner	Loser	Minimum Winner Tally	Maximum Loser Tally	Difficulty
Alice NEB Diego	Alice	Diego	4000	4500	–
Bob NEB Diego	Bob	Diego	1000	3500	–
Chuan NEB Diego	Chuan	Diego	5000	7500	–

(c) NEB assertions to show that Diego cannot be the winner.

Table 1.1: Potential NEB assertions to show that (a) Alice, (b) Bob, and (c) Diego cannot be the last candidate standing. An NEB assertion between a winner w and a loser l can be formed when the maximum tally of the loser (l) is greater than the minimum tally of the winner (w). A '–' in the Difficulty column means that this property does not hold for the associated assertion, and it *cannot* be formed.

RAIRE will now move its attention to node 2 (Bob). Table 1.1b shows that we can form *one* NEB assertion stating that *Chuan NEB Bob* (i.e., Chuan will always have more votes than Bob, irrespective of who is eliminated prior). The maximum tally Bob can have while Chuan is still continuing is 1000 votes. Bob will have the 1000 (B, C, D) ballots. Chuan's minimum tally is his fist preference tally of 5000 votes. RAIRE will assign the assertion *Chuan NEB Bob* a difficulty of 1/((5000-1000)/13500) = 3.375.

Based on the above, RAIRE will assign node 2 (Bob) a difficulty of 3.375, and keep a record of the best assertion it found to rule out the outcome where Bob wins (*Chuan NEB Bob*).

RAIRE will now move its attention to node 3 (Diego). Table 1.1c shows that RAIRE cannot form any NEB assertions at this stage to show that Diego could not be the eventual winner. RAIRE will assign node 3 (Diego) a difficulty of infinity. Figure 1.2 shows the current state of our alternate outcomes trees.

One approach to creating a valid audit is to walk down each branch of our set of alternate outcome trees until we find a node that we can attack with an assertion. In Figure 1.2, we have already found a way to attack all branches that end with Bob as the winner. We would then simply have to explore possible children of Alice and Diego, and their children, until we have a way of ruling out each branch. We can

First Level of Alternate Outcomes Tree Set – Evaluated

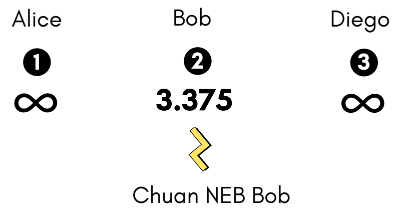

Figure 1.2: Evaluation of the first level of nodes in our alternate outcomes tree set.

then collect all the assertions we have used to form our audit. We have no way of knowing, however, if this is a *good* audit in terms of the number of ballots auditors will have to check. There may be a much cheaper way of verifying the announced outcome. The method that RAIRE employs is designed to not only *find* a set of assertions that verify that the announced winner won, but to *guarantee* that there is no cheaper audit according to the approach chosen to estimate assertion difficulty. The question is how do we know what the cost of the optimal audit is going to be without looking at all ways of ruling out each branch with an assertion? RAIRE does this by maintaining what it calls a 'lower bound' on the cost of any valid audit. Initially, this lower bound is set to zero, as RAIRE has no information to indicate what the optimal cost of an audit is going to be. RAIRE can only update this lower bound when it considers an entire branch in our set of alternate outcomes trees, and establishes the cheapest assertion to rule out the outcome that it represents. RAIRE knows that it needs to rule this outcome out somehow, and that the overall cost of the audit must be *at least* equal or greater than the cost of this 'cheapest assertion'. At this point, RAIRE will increase its lower bound on overall audit cost to the cost of this asertion. Whenever RAIRE has a way of ruling out each branch with an assertion whose cost is less than or equal to this bound, we know that we have a *least-cost audit*. We also want to minimize the number of complete branches RAIRE has to look at before it can be sure that it knows what the optimal audit cost is going to be. We can achieve this by getting RAIRE to explore our set of alternate outcomes trees more intelligently, trying to find the branch that will be hardest to rule out as early as possible.

RAIRE explores the set of alternate outcomes trees through a process called *node expansion*.

Consider Figure 1.2. RAIRE will now look at all nodes in our trees that do not currently have any children, but are not complete outcomes. We call these nodes *expandable*. In Figure 1.2, nodes 1, 2, and 3 are all expandable. Of these expandable nodes, RAIRE selects the node with the highest difficultly estimate. In this case, RAIRE could select either node 1 (Alice) or node 3 (Diego), but will select node 1 (Alice) as it was 'created' first. RAIRE will add a child under Alice in our tree for every possible candidate that could be her runner-up. We call this process *expansion*. Bob, Chuan, and Diego are the three candidates that could be Alice's runner-up. Three new nodes are added to our tree (Figure 1.3), one for each of these candidates. Node 4 (Bob, Alice), for example, represents an outcome that ends with Bob as the runner-up and Alice as the winner.

Expansion of Node 1

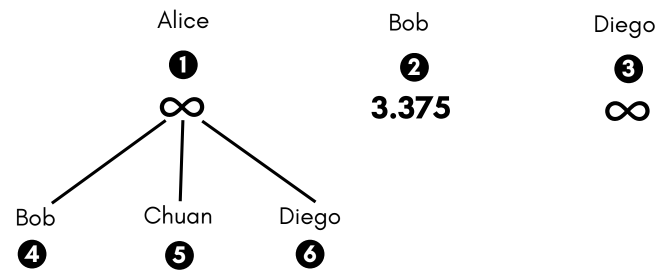

Figure 1.3: 'Expansion' of node 1, creating three new nodes in our alternate outcomes tree set.

RAIRE will now look at nodes 4, 5, and 6, in turn, and search for assertions that would rule out the outcomes that they represent. When we talk about a node that is not at the top level of one of our alternate outcomes trees, we will often refer to the *elimination order* or set of orders that it represents. Node 4, for example, represents the set of outcomes in which Bob is the runner-up, and Alice is the winner. We will often describe such nodes with a sequence, e.g., (Bob, Alice). When these sequences do not contain all candidates, they represent the *tail end* of an elimination order. Here, the sequence (Bob, Alice) does not mean that Bob is eliminated first, and Alice is eliminated next – it means that Alice is the last candidate standing, and Bob is the runner-up. The sequence (Chuan, Bob, Alice) refers to the set of outcomes where Chuan is eliminated second, Bob is eliminated next, and Alice is the ultimate winner. Finally, the sequence (Diego, Chuan, Bob, Alice) represents one specific elimination order where Diego is eliminated first, Chuan next, Bob as the runner-up, and Alice as the winner.

Now that we are looking at nodes that make statements about multiple candidates (i.e., they describe an elimination order 'tail' sequence with multiple candidates in it) RAIRE will consider both NEB and NEN assertions when trying to rule out the set of outcomes the nodes represent. For node 4, RAIRE will first consider whether it can show that one of the candidates not present in the branch between node 1 and 4, Chuan and Diego, could not be eliminated before Bob. If so, then RAIRE may form the assertions *Chuan NEB Bob* or *Diego NEB Bob*. Either would rule out the outcome where Bob is the runner-up and Alice is the winner. Table 1.1b tells us that we can form *Chuan NEB Bob*, of cost 3.375, but not *Diego NEB Bob*. Next, RAIRE will consider whether, at the point where only Alice and Bob remain, Bob actually has more votes than Alice. In this context, Alice has 5500 votes and Bob has 6000. So, we can form *NEN: Bob > Alice if only {Bob, Alice} remain* with a cost of $1/(500/13500) = 27$. The NEB assertion is a cheaper way of attacking the outcome, and so we assign it, and its cost, to node 4. This is shown in Figure 1.4.

This process of looking for NEB and NEN assertions to attack a node is called *evaluation*. To summarise, evaluation involves: looking for NEB assertions to attack the node's outcome; looking for NEN assertions to attack the node's outcome; selecting the least cost of these assertions (if any were found); and as-

Listing 1.1: Process RAIRE uses to evaluate a node

Let N denote an arbitrary node representing the tail end of a possible elimination order:

$$\pi = [c,\ldots,w]$$

where $c,\ldots,w$ is a sequence of candidates, and w the eventual winner.

RAIRE applies a specific process for generating assertions to attack this node.

1. RAIRE will look at the last candidate that was added to the front of the node's elimination sequence, c. For all candidates o, that are *not mentioned* in this elimination sequence, RAIRE will determine if it can form the assertion *o NEB c*. If this assertion can be formed, then it provides one way of ruling out the outcome that N represents, as it is not possible to have an outcome where o appears before c in the elimination sequence. RAIRE will record the difficulty of these NEB assertions.

2. Next, RAIRE will look at how many votes each candidate in π has in the context where all candidates *not mentioned* in π have been eliminated (i.e., π represents the set of candidates still continuing). If c has *more* votes than some other continuing candidate, o', in π, then we can form the assertion *NEN: $c > o'$ if only $\{\pi\}$ remain*.

3. RAIRE will choose the least difficult of the NEB and NEN assertions it was able to form, and assign this difficulty, and the assertion itself, to N. If RAIRE was not able to form any assertions, N will be assigned a difficulty of infinity.

signing the chosen assertion to the node with its associated difficulty, or assigning a difficulty of infinity if no assertions were found. The process of evaluating a node is specified more formally in Listing 1.1.

RAIRE will repeat this evaluation process, as described in Listing 1.1, for nodes 5 (Chuan, Alice), and 6 (Diego, Alice). For node 5, RAIRE first considers whether it can show that Bob or Diego cannot be eliminated before Chuan (Step 1 of Listing 1.1). Neither *Bob NEB Chuan* or *Diego NEB Chuan* can be formed. For *Bob NEB Chuan*, Bob's minimum tally is 1000, and Chuan's maximum tally, while Bob remains as a continuing candidate, is 5000. For *Diego NEB Chuan*, Diego's minimum tally is 1500, and Chuan's maximum tally while Diego remains is 6000. RAIRE next considers whether Chuan has more votes than Alice in the context where only Alice and Chuan remain (Step 2 of Listing 1.1). In fact, Chuan would have 6000 votes and Alice 5500. So, we can form the assertion *NEN: Chuan > Alice if only {Chuan,Alice} remain*, of cost $1/(500/13500) = 27$. Node 5 will be assigned a difficulty of 27 (Step 3 of Listing 1.1).

Repeating this process for node 6 (Diego, Alice), RAIRE considers whether it can show that either Chuan or Bob cannot be eliminated before Diego. We know from Table 1.1c that neither *Chuan NEB Diego* or *Bob NEB Diego* can be formed. RAIRE next considers whether Diego has more votes than Alice in the

Expansion of Node 1 – Evaluation

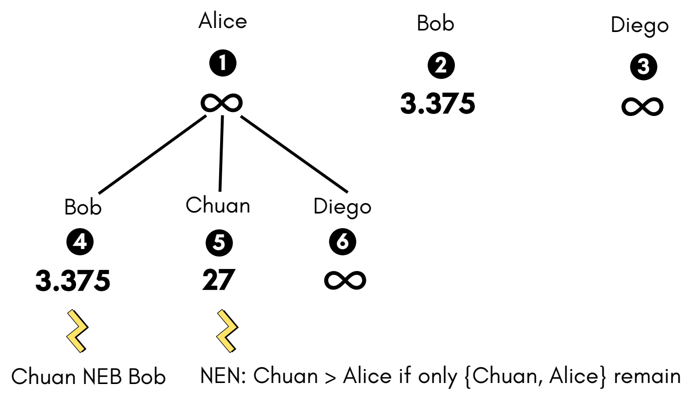

Figure 1.4: Evaluation of nodes 4-6 in our alternate outcomes tree set.

Expansion of Node 3 – Evaluated

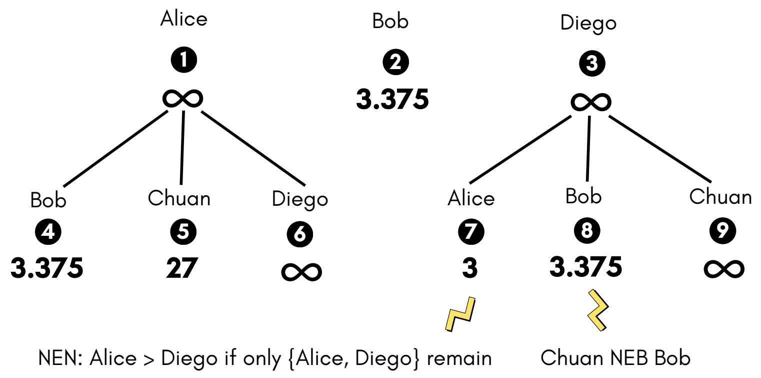

Figure 1.5: Alternate outcomes trees after the expansion of node 3 (Diego) and evaluation.

context where only they remain standing. In this context, Alice has 9000 votes and Diego has 4500. So, we cannot form the assertion *NEN: Diego > Alice if only {Diego,Alice} remain* and node 6 is assigned a difficulty of infinity. Figure 1.4 shows the current state of our set of alternate outcomes trees.

We are again in a context where all childless nodes in our trees have been evaluated. RAIRE now looks at all nodes in our tree set that do not currently have children, but are not complete outcomes–the set of *expandable* nodes–and selects a node with the highest difficultly estimate. RAIRE will select node 3 (Diego) with its difficulty of infinity. It could have selected node 6 (Diego, Alice), but node 3 was created earlier and RAIRE uses this as its tie-breaking rule. RAIRE will add a child under Diego for every possible candidate that could be his runner-up. These candidates are Alice, Bob, and Chuan. RAIRE will evaluate these new nodes (7-9) following the process in Listing 1.1. The result is shown in Figure 1.5.

Expansion of Node 6 – Evaluated

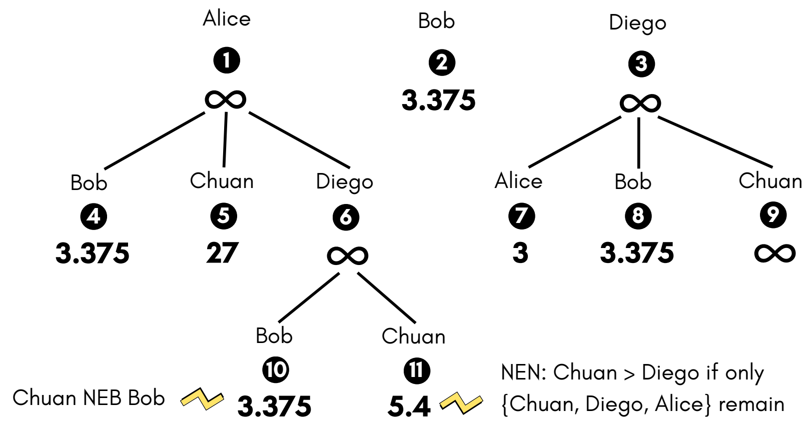

Figure 1.6: Alternate outcomes tree set after expansion of node 6.

Expansion of Node 9 – Evaluated

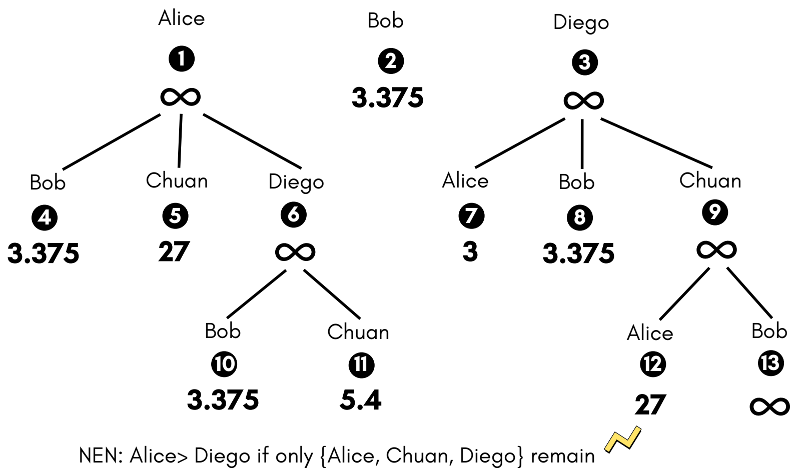

Figure 1.7: Alternate outcomes tree set after expansion of node 9.

RAIRE continues its process of selecting a node with the highest assigned difficulty, expanding it by adding a child for every candidate not mentioned in the node's elimination sequence, and evaluating the newly created children. Figure 1.6 shows our set of alternate outcomes trees after RAIRE has expanded node 6 (Diego, Alice), with its difficulty of infinity, and has evaluated each of its children.

RAIRE will next expand node 9 (Chuan, Diego), with its difficulty of infinity, and evaluate its children. Figure 1.7 shows the resulting state of our set of alternate outcome trees.

Expansion of Node 13 – Evaluated

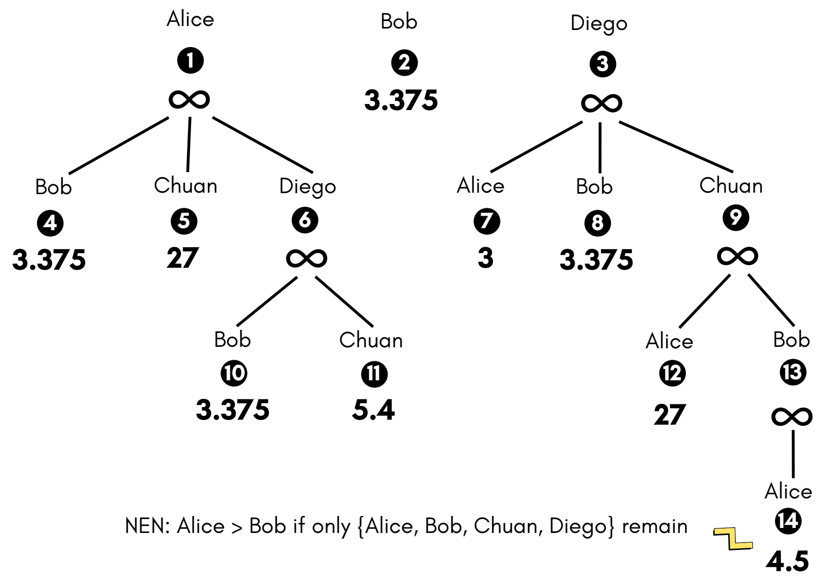

Figure 1.8: Alternate outcomes tree set after expansion of node 13.

RAIRE shifts its attention to node 13 (Bob, Chuan, Diego), as this node has the highest difficulty. This node has only one child, which is added to our trees as node 14 (Alice, Bob, Chuan, Diego). Only one assertion can be formed to attack this node: *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*. Alice has a tally of 4000 ballots in the context where no candidate has yet been eliminated, and Bob has a tally of 1000 ballots. The assertion is assigned a difficulty of $1/(3000/13500) = 4.5$. Figure 1.8 shows the current state of our set of alternate outcomes trees.

Are we done yet?

RAIRE has now found enough true assertions to rule out all possible winners other than the announced winner (Chuan). We could (but won't) stop here and conduct an audit using all the assertions RAIRE has so far formed. This would be a valid RLA. However, it might be very inefficient because some elimination orders may be more easily ruled out using assertions that attack them closer to the leaves. RAIRE therefore does not *immediately* include any of these assertions in the audit. Instead, it continues to examine more of the elimination trees, starting with the most expensive nodes, and expanding them to see whether these branches can be attacked with assertions that are easier to audit. Assertions are added to the audit only when RAIRE has calculated that either

- the full elimination sequence cannot be ruled out with a lower-cost assertion, or
- the assertion is no harder to audit than other assertions already in the audit.

Listing 1.2: Establishing a lower bound on the audit cost

RAIRE applies a key principle to avoid having to explore entire alternate outcomes trees. RAIRE's aim is to form a set of least-cost assertions while minimising how much of each alternate outcomes tree we need to explore. Visiting every node in these trees, for real-world elections, would be intractable. The method needs to return a set of assertions, or indicate that a full manual count is required, in a reasonable amount of time. The key mechanism RAIRE uses to do this is to maintain, and continually update, a *lower bound* on the difficulty of our audit.

RAIRE starts with a lower bound of zero, and revises this bound whenever we reach a *leaf* in our exploration of the trees. Recall that at a leaf, it becomes clear which node, N, is the *weakest point* in its branch. If the cost of the assertion used to attack the branch at N is C, then our audit must have a cost of at least C, as we have found the least cost way of ruling out all outcomes (branches of the tree) that pass through this weakest point, N. If C is *higher* than our current lower bound, we revise the bound to be equal to C. We then remove node N, and all of its descendants, from our alternative outcomes trees, and add the assertion that attacks N to our audit. We then look at all of the branches that remain in our trees, and whether their current weakest points have attacking assertions with costs that are less than or equal to C.

We repeat this process of adding these assertions to our audit, and removing these weakest point nodes, and their descendants, from our trees.

1.1.1 Building the audit from the assertions that are easiest to audit

Node 14 (Alice, Bob, Chuan, Diego) is a *leaf* node as it represents a complete outcome. Listing 1.2 describes how RAIRE uses leaf node evaluations to avoid having to explore the entire alternate outcomes tree.

Now that RAIRE has created and assigned a difficulty to a *leaf* node, it has a better idea of how difficult the overall audit will be. RAIRE knows that it will have to rule out the alternate outcome with the elimination sequence Alice, Bob, Chuan, Diego somehow. It can only do this by attacking one of the nodes on the *branch* starting at the top of the tree and ending at node 14. This branch contains the nodes 3 (Diego), 9 (Chuan, Diego), 13 (Bob, Chuan, Diego), and 14 (Alice, Bob, Chuan, Diego). The cheapest, and in this case only, way to do this is to attack the leaf (node 14) with the assertion *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*. RAIRE adds this assertion to our audit, which now has a cost of 4.5.

Our Audit *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*
Cost: 4.5

RAIRE then looks at any additional outcome that it knows can be ruled out with a cost of 4.5 or less. The outcomes represented by nodes 2 (Bob), 4 (Bob, Alice), 7 (Alice, Diego), 8 (Bob, Diego), and 10 (Bob, Diego, Alice) can all be ruled out with assertions that have a cost less than 4.5. RAIRE will now add the assertions used to attack these nodes to our audit.

Our Audit *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*
Chuan NEB Bob
NEN: Alice > Diego if only {Alice,Diego} remain

Cost: 4.5

Once these alternate outcomes have been ruled out, RAIRE can *prune* them from our set of alternate outcomes trees. *Pruning* essentially means that RAIRE removes those nodes from our trees. When all descendants of a node have been removed, we also remove that node from our set of trees. After this first pruning, Figure 1.9 shows the state of our set of alternate outcomes trees.

RAIRE will now take the slimmed down set of alternate outcomes trees, and continue its process of expanding the node with the highest difficulty, adding children to the trees, and evaluating them. Looking at the trees in Figure 1.9, RAIRE will next expand node 5 (Chuan, Alice), creating nodes 15 (Bob, Chuan, Alice) and 16 (Diego, Chuan, Alice). Both nodes will be assigned a difficulty of infinity (Figure 1.10).

Alternate Outcomes Tree Set – First Pruning

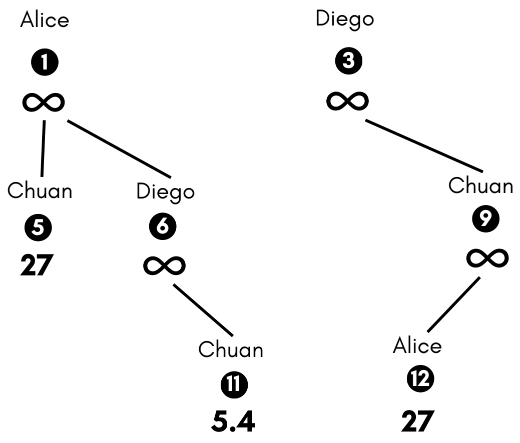

Figure 1.9: Alternate outcome tree set after their first pruning.

Expanding node 15 (Bob, Chuan, Alice), with difficulty infinity, adds one new node, node 17 (Diego, Bob, Chuan, Alice), to our set of trees. The best assertion RAIRE can create to attack node 17 is *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain*, with difficulty 5.4. Figure 1.10 shows the state of our alternate outcomes trees at this point. Node 17 is a leaf node representing an outcome with the elimination

Alternate Outcomes Tree Set – Expansion of Node 15

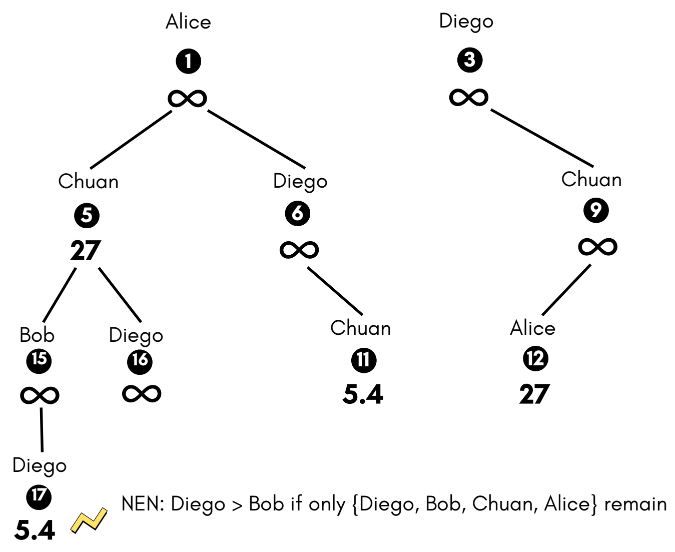

Figure 1.10: Alternate outcome tree set after expansion of node 15.

sequence Diego, Bob, Chuan, Alice. RAIRE now needs to rule out this outcome, somewhere on the branch involving nodes 1 (Alice), 5 (Chuan, Alice), 15 (Bob, Chuan, Alice), and 17 (Diego, Bob, Chuan, Alice). It could do this with the assertion *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain*, attacking node 17, which would result in nodes 17 and 15 being removed from our trees at a cost of 5.4.

Alternatively, RAIRE could use *NEN: Chuan > Alice if only {Chuan,Alice} remain* to attack the branch leading to node 17 at node 5 (Chuan, Alice), at a cost of 27. This would result in node 5, and all of its descendants (nodes 15-17), being removed. The attack at node 17 is cheaper, however, and RAIRE will choose the cheaper option. The assertion *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain* is added to our audit, increasing its cost to 5.4, and nodes 17 and 15 are removed from out tree set.

Our Audit	*NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*
	Chuan NEB Bob
	NEN: Alice > Diego if only {Alice,Diego} remain
	NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain
Cost: 5.4	

RAIRE increases its lower bound on expected audit cost to 5.4, and looks at what alternate outcomes it can rule out with a cost of 5.4 or less. We can rule out node 11, the outcome (Chuan, Diego, Alice), with an assertion of cost 5.4. This assertion, *NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain*, is added to our audit and nodes 11 and 6 are removed from our trees.

Alternate Outcomes Tree Set – Expansion of Node 16

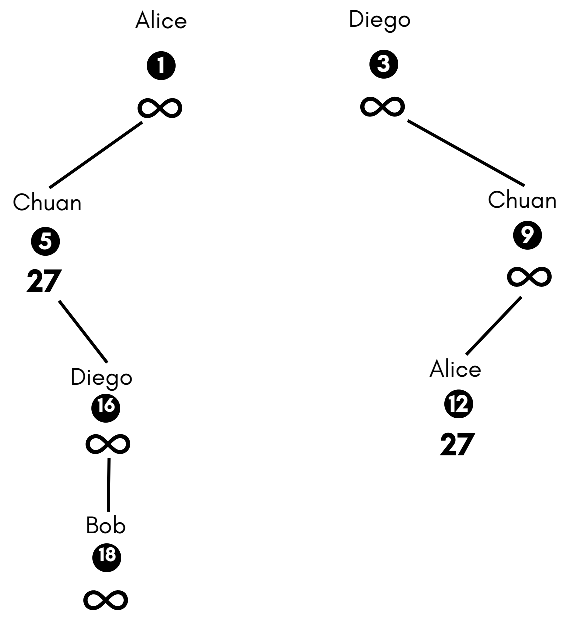

Figure 1.11: Alternate outcome tree set after expansion of node 16.

RAIRE then finds another node to expand. It selects node 16 (Diego, Chuan, Alice) as it has the highest difficulty (of infinity). Node 18 (Bob, Diego, Chuan, Alice) is added to the tree, as shown in Figure 1.11. It will be assigned a difficulty of infinity as we cannot show that Bob has more votes than any of the other candidates at the point where no one has been eliminated. We know that we need to rule this outcome out, and the only way to do so is to attack node 5 (Chuan, Alice) with the assertion *NEN: Chuan > Alice if only {Chuan,Alice} remain*, at cost 27. This assertion is added to our audit.

Our Audit *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*
Chuan NEB Bob
NEN: Alice > Diego if only {Alice,Diego} remain
NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain
NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain
NEN: Chuan > Alice if only {Chuan,Alice} remain

Cost: 27

RAIRE increases its lower bound on expected audit cost to 27, and looks at what alternate outcomes it can rule out with a cost of 27 or less. It can rule out the single remaining outcome–the branch starting at node 3 (Diego) an ending at node 12 (Alice, Chuan, Diego)–by including the assertion that attacks node

12. This assertion is *NEN: Alice > Diego if only {Alice,Chuan,Diego} remain*. Adding this assertion to our audit gives us a set of assertions that rule out all outcomes in which the reported winner did not win.

Our Audit *NEN: Alice > Bob if only {Alice,Bob,Chuan,Diego} remain*
Chuan NEB Bob
NEN: Alice > Diego if only {Alice,Diego} remain
NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain
NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain
NEN: Chuan > Alice if only {Chuan,Alice} remain
NEN: Alice > Diego if only {Alice,Chuan,Diego} remain

Cost: 27

In this example, we have a redundant assertion: *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain*, of cost 5.4. At the time this assertion was added to our audit, we were trying to see if we could avoid the more expensive assertion, *NEN: Chuan > Alice if only {Chuan,Alice} remain*, costing 27. The latter assertion rules out all outcomes that end with Chuan as the runner-up and Alice as the winner. The former assertion rules out a specific outcome, one where Diego is eliminated first, Bob next, with Chuan as the runner-up, and Alice as the winner. Once *NEN: Chuan > Alice if only {Chuan,Alice} remain* has been added to our audit, there is no need for *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain* to remain as the outcome it was targeting is also ruled out by the more costly assertion.

Refining our Assertion Set RAIRE currently uses several rules for detecting certain kinds of redundancies in an assertion set. These rules are described below. For each assertion A in our audit, we keep track of the partial or complete outcomes O_A that it is being used to rule out. We may be using one assertion to rule out multiple partial or complete outcomes.

1. Each NEN assertion is designed to rule out outcomes where a specific candidate w is eliminated at an elimination step when candidates $\mathcal{S}$ remain standing. Consider an NEB assertion with winner w and loser l. If w is eliminated before l in all outcomes that the NEN assertion rules out, then the NEB makes the NEN assertion redundant. For example, let's consider the following two assertions:

   Chuan NEB Bob
   NEN: Chuan > Bob if only {Chuan,Bob,Alice} remain

   The NEB rules out all outcomes in which Chuan is eliminated while Bob remains. The NEN rules out a smaller set of outcomes where Chuan is eliminated at the point where Chuan, Bob, and Alice remain. We can see that this smaller set of outcomes is a subset of those that the NEB rules out.

Consider a different NEN assertion: *NEN: Chuan > Alice if only {Chuan,Bob,Alice} remain*. This is also made redundant by the assertion *Chuan NEB Bob*.[2]

We say that the assertion $A =$ *NEN: $w > l$ if only $\{\mathcal{S}\}$ remain* is made redundant by an w' *NEB* l' if in each partial or complete elimination order $\pi \in O_A$ it is implied that w' is eliminated before l'.

2. The NEN assertion A_1 makes the NEN assertion A_2 redundant if for all outcomes $\pi \in O_{A_2}$ there is an outcome being ruled out by A_1, $\pi' \in O_{A_1}$, where π' is a *suffix* of π. Let's consider the redundant assertion that was found in our example audit:

   $A_1 =$ *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain*

   This assertion is being used to rule out the outcome (Diego, Bob, Chuan, Alice). We later added the assertion:

   $A_2 =$ *NEN: Chuan > Alice if only {Chuan,Alice} remain*

   This assertion is being used to rule out the partial outcome (Chuan, Alice) representing all outcomes that end with Chuan as the runner-up and Alice as the winner. We can see that for each outcome π being ruled out by A_1, there is an outcome being ruled out by A_2, π', where π' is a suffix of π. Here, $\pi' =$ (Chuan, Alice) is a suffix of $\pi =$ (Diego, Bob, Chuan, Alice).

These rules consider how *one* assertion may make another assertion redundant. It is possible for a set of NEN assertions to make another of our NEN assertions redundant. More sophisticated techniques are available to filter redundant assertions of this kind from our audit.

Finding the least-cost audit Recall that we did not want to simply walk down each branch of our alternate tree set until we found a point that we could attack with an assertion. This approach would lead to a valid audit, by collecting all the assertions used to attack each branch, but this audit may not be the cheapest one. In our running example, it just happens that this approach *would have* resulted in a least-cost audit, with a total cost of 27. Let's look at a different example.

[2] It is not *logically* redundant, but it is redundant for the purposes of auditing, because we use the assertion *NEN: Chuan > Alice if only {Chuan,Bob,Alice only to rule out Chuan getting eliminated when Chuan,Bob and Alice} remain* are continuing. The assertion *Chuan NEB Bob* also implies that Chuan cannot be eliminated at that point.

Example 2. *Suppose there are 3 candidates: Alice, Bob, and Chuan. The votes are as follows:*

Preferences	***Count***
(A, B, C)	*5000*
(A, C)	*5000*
(B, C, A)	*5000*
(B)	*6000*
(C, B)	*10000*
(C)	*10000*

Alice, Bob, and Chuan initially have 10000, 11000, and 20000 votes, respectively. Alice has the smallest tally and is eliminated first, with 5000 of her votes transferred to Bob and 5000 to Chuan. Bob and Chuan now have 16000 and 25000 votes, respectively. Chuan has more votes than Bob and is declared the winner.

Now consider the first level of nodes in our alternate outcomes trees for this example, as shown in Figure 1.12. There are just two, one representing all outcomes that end in Alice as the winner, and one that represents all outcomes that end in Bob as the winner. The best assertions we can form to rule out these two nodes are *Chuan NEB Alice* with a cost of 1/((20000-10000)/41000) = 4.1, and *Chuan NEB Bob* with a cost of 1/((20000-16000)/41000) = 10.25.

First Level Evaluations

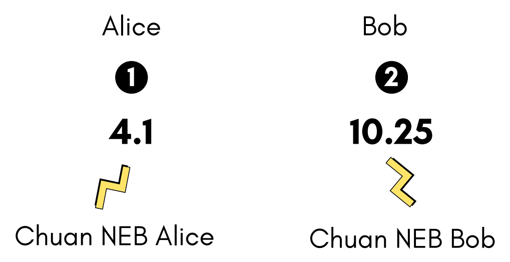

Figure 1.12: First level of our alternate outcomes trees for Example 2.

These two NEB assertions form a valid audit to rule out all outcomes in which the reported winner, Chuan, does not win. This audit will have a total cost of 10.25. Is this the cheapest audit we could form? Consider now the second level of nodes in our alternate outcomes trees, as shown in Figure 1.13.

We can rule out node 3 (Bob, Alice) with the assertion *NEN: Bob > Alice if only {Bob,Alice} remain* with cost 1/((21000-10000)/41000) = 3.7. Node 4 (Chuan, Alice) can be ruled out with the assertion *NEN: Chuan > Alice if only {Chuan,Alice} remain* with cost 1/((25000-10000)/41000) = 2.7. Nodes 5 (Alice, Bob) and 6 (Chuan, Bob) can be ruled out with the assertions *Chuan NEB Alice* of cost 4.1 and *NEN: Chuan > Bob if only {Chuan,Bob} remain* with cost 1/((25000-16000)/41000) = 4.6.

Second Level Evaluations

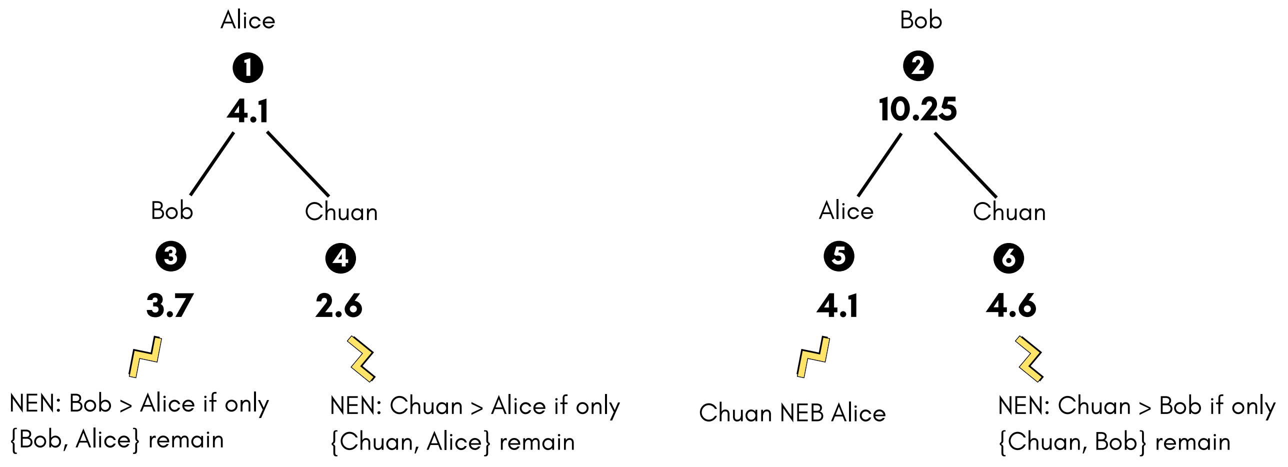
Alice
1
4.1
Bob
3
3.7
NEN: Bob > Alice if only {Bob, Alice} remain
Chuan
4
2.6
NEN: Chuan > Alice if only {Chuan, Alice} remain
Bob
2
10.25
Alice
5
4.1
Chuan NEB Alice
Chuan
6
4.6
NEN: Chuan > Bob if only {Chuan, Bob} remain

Figure 1.13: Second level of our alternate outcomes trees for Example 2.

We now know that we can form a valid audit with a total cost of 4.6. This audit includes *Chuan NEB Alice*, used to attack nodes 1 and 5, and *NEN: Chuan > Bob if only {Chuan,Bob} remain*, used to attack node 6. In fact, 4.6 is the optimal cost for this example–we cannot form a valid audit that is cheaper than this.

Chapter 2

Optimizing RAIRE

Recall that RAIRE is designed to find a least-cost set of assertions that, if true, demonstrate that the announced winner won. It not only has to find such a least-cost set, but it has to prove that there is no other set with a cheaper cost. It does this by maintaining a lower bound on the cost of the optimal audit. This lower bound starts at zero, and can only be updated when RAIRE visits a leaf (complete outcome) in our collection of alternate outcomes trees. It is only when RAIRE fully explores a branch that it knows how the branch can be ruled out in the cheapest way. Each time RAIRE visits a leaf, it looks at all the ways in which the branch leading to that leaf can be ruled out with an assertion. It knows that *any* valid audit must include one of these assertions. So, RAIRE will include the cheapest of these in its audit, and update its lower bound on the cost of an optimal (least-cost) valid audit accordingly.

If we are able to update this lower bound more quickly, RAIRE will most likely accept ways of attacking branches at earlier points in the trees without needing to explore further. For example, consider a situation where we know we can attack all outcomes that end with a specific alternate winner, Alice, with an an assertion that costs 15. If our lower bound is 5, we may need to explore children of Alice, and potentially their children, to find ways of ruling out Alice as a winner that fit within our current bound of 15. If our lower bound was 20, we could accept the known way of ruling out Alice as a winner – the assertion that costs 15 – without exploring descendants of Alice in our tree.

We have incorporated a key optimization within RAIRE – called *diving* – that specifically aims to update our lower bound more quickly, and to reduce the number of nodes RAIRE has to evaluate in order to form a least-cost assertion set. As we can only update this lower bound when RAIRE reaches a leaf, the goal of diving is to rapidly descend to a leaf in our alternate outcomes trees in a way that is likely to yield the largest increase in our lower bound. We can think of diving as running down a particular branch of the tree – one that we think will be represent the hardest outcome to rule out with an assertion. We make the assumption that the hardest alternate elimination sequence to rule out is one that is equal to the reported sequence but with the winner and runner-up swapped.

Diving comes into play when RAIRE *expands* a node. Let's cast our minds back to Figure 1.2, where RAIRE had added the first level of nodes to our set of alternate outcome trees, and evaluated them. In this first level, we had node 1 (Alice) with a difficulty of infinity, node 2 (Bob) with a difficulty of 3.375, and node 3 (Diego) with a difficulty of infinity. Node 1 was first selected for expansion.

Before we expand Node 1 fully, we perform a *dive*.

Figure 2.1 shows the result of this dive. We take our to-be-expanded node and consider only *one* of its children. We choose a runner up for Alice that allows us to most closely replicate the reported outcome. Our assumption is that this will increase our changes of creating an elimination sequence that is harder to rule out than others. Chuan lasts longer than Bob and Diego in the reported outcome, and so we add the node (Chuan, Alice) to our tree as node 4. We then evaluate the added node. The best assertion we can find to attack node 4 is *NEN: Chuan > Alice if only {Chuan,Alice} remain* with a cost of 27. We then take the node we just added (node 4) and consider *one* of its children. Diego lasts longer than Bob, and so we add the node (Diego, Chuan, Alice) to our tree as node 5. When evaluating node 5, RAIRE finds it cannot form any assertions to attack it, so it is assigned a difficulty of infinity. We then take the node we just added (node 5), and consider *one* of its children. There is only one candidate left, and we add the node (Bob, Diego, Chuan, Alice) to our tree set as node 6. When RAIRE evaluates this node, it finds it cannot find an assertion to attack it. Diving continues like this until we reach a leaf, or a node that can be attacked with an assertion whose cost is less than or equal to the current lower bound on audit cost.

Alternate Outcomes Tree Set – First Dive

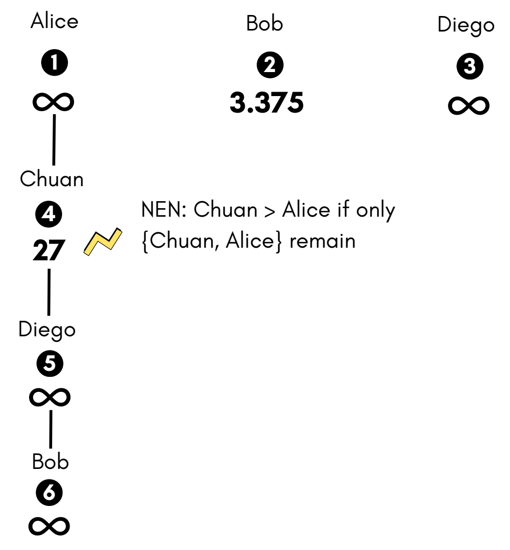

Figure 2.1: Alternate outcome tree set after first dive when expanding node 1.

Now that we have reached a leaf node (node 6), we need to rule out the outcome represented by the branch starting at node 1 (Alice) and ending at node 6 (Bob, Diego, Chuan, Alice). The only way we can

Alternate Outcomes Tree Set – Expansion of Node 1

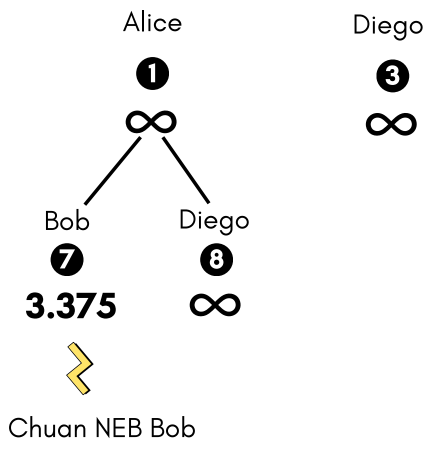

Figure 2.2: Alternate outcome tree set after expansion of node 1.

do this is with the assertion currently attacking node 4 (Chuan, Alice). This assertion is *NEN: Chuan > Alice if only {Chuan,Alice} remain* with cost 27. We add this assertion to our audit, remove nodes 4, 5 and 6 from our trees, and increase our lower bound on expected audit cost to 27. Now that RAIRE has a non-zero lower bound on expected audit cost, it looks at what additional outcomes it already knows how to rule out with a cost of 27 or less. There is one such set of outcomes – those that end with Bob as the winner. We add *Chuan NEB Bob* to our audit and remove node 2 (Bob) from our set of trees.

Our Audit *NEN: Chuan > Alice if only {Chuan,Alice} remain*
Chuan NEB Bob

Cost: 27

RAIRE will then complete the expansion of node 1, adding and evaluating its two additional children. Figure 2.2 shows the result. As we an attack node 7 with a cost that is less than our current lower bound on the cost of the audit (27), with the assertion *Chuan NEB Bob*, we remove node 7 from our trees. (As this NEB is already in our audit, we do not need to replicate it). RAIRE will then select node 3 (Diego) for expansion, first completing a dive until it reaches a node that is either a leaf or that can be attacked with an assertion that costs less than or equal to 27. Figure 2.3 shows the result.

Figure 2.4 shows the result of the completed expansion of node 3 (Diego). Its remaining two children, not formed during the dive, have been added to the tree and evaluated. Both can be attacked with assertions that have a cost less than our current audit cost. So, the assertions attacking these nodes are added to our audit, and the nodes removed from our tree.

Alternate Outcomes Tree Set – Diving from Node 3

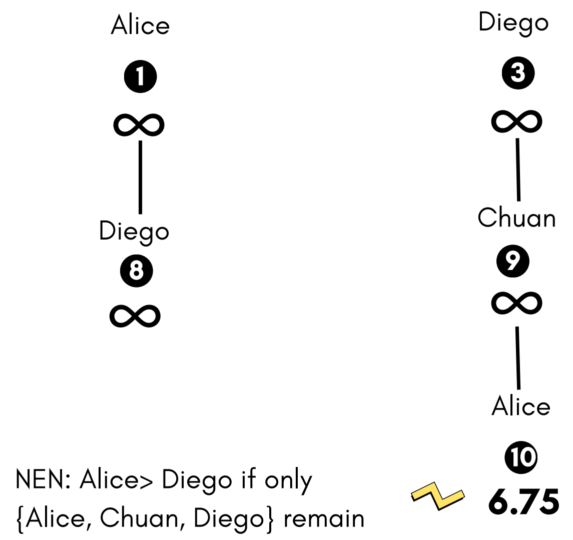

Figure 2.3: Alternate outcome tree set after diving from node 3.

Alternate Outcomes Tree Set – Expansion of Node 3

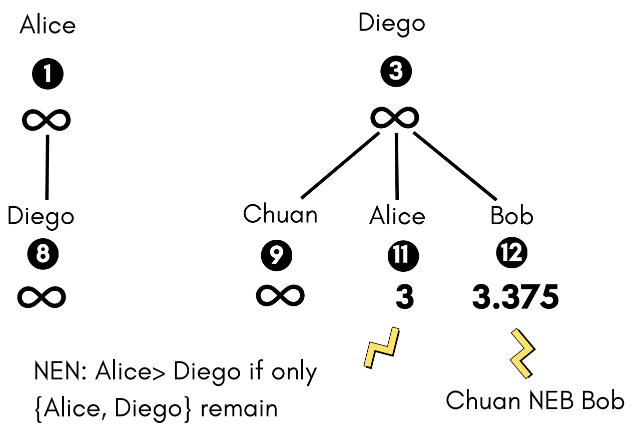

Figure 2.4: Alternate outcome tree set after expansion of node 3.

Our Audit *NEN: Chuan > Alice if only {Chuan,Alice} remain*
Chuan NEB Bob
NEN: Alice > Diego if only {Alice,Chuan,Diego} remain
NEN: Alice > Diego if only {Alice,Diego} remain

Cost: 27

Alternate Outcomes Tree Set – Diving from Node 8

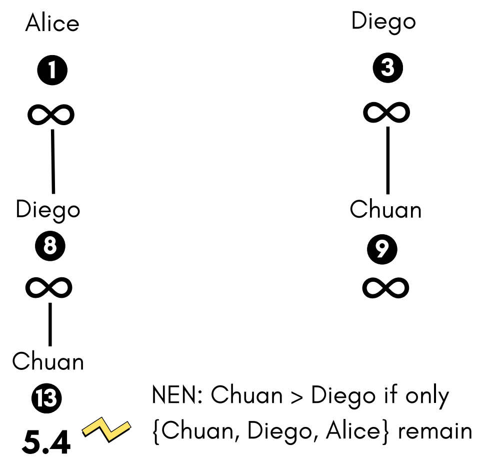

Figure 2.5: Alternate outcome tree set after diving from node 8.

Alternate Outcomes Tree Set – Expansion of Node 8

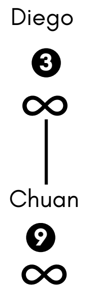

Figure 2.6: Alternate outcome tree set after the expansion of node 8.

RAIRE then selects node 8 (Diego, Alice) to expand. First, it will perform a dive, adding a child (node 13) representing the outcome (Chuan, Diego, Alice). When evaluated, this new node can be attacked by an assertion with a cost of 5.4 (*NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain*). This cost is less than our current audit cost, and so diving stops. Figure 2.5 shows the state of our alternate outcome tree set at this point. We can rule out node 13, adding the attacking assertion to our audit. When RAIRE completes the expansion of node 8, adding the one remaining child (Bob, Diego, Alice), it finds this child can be ruled out with the assertion *Chuan NEB Bob* at a cost of 3.375. This is within our current audit cost, and incidentally the assertion is already in our audit, and so node 13 (along with node 8, and node 1) is removed from our tree. All outcomes in which Alice wins have now been ruled out.

Figure 2.6 shows our much simplified alternative outcome tree set.

Alternate Outcomes Tree Set – Diego ruled out as a Winner

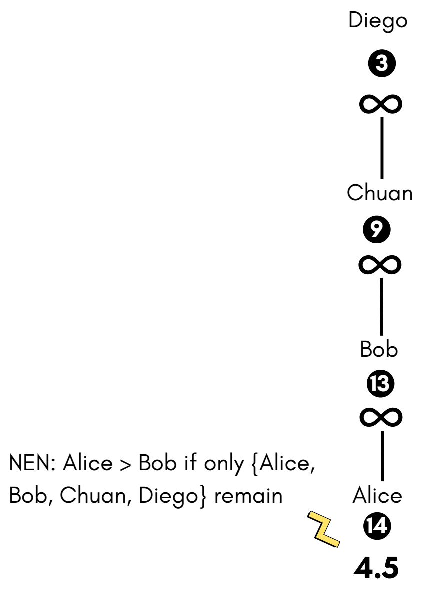

Figure 2.7: Alternate outcome tree set after expansion of nodes 9 and 13.

Our Audit	*NEN: Chuan > Alice if only {Chuan,Alice} remain*
	Chuan NEB Bob
	NEN: Alice > Diego if only {Alice,Chuan,Diego} remain
	NEN: Alice > Diego if only {Alice,Diego} remain
	NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain

Cost: 27

RAIRE will now select node 9 (Chuan, Diego) for expansion. This node has already been involved in a dive – that is how it was added to the tree – and so we don't perform the dive again. We simply consider the children of node 9 that were not added to our tree in the earlier dive from node 3. There is only one such child. We add a node for the outcome (Bob, Chuan, Diego) to our tree as node 13. It will be assigned a difficulty of infinity, however, as we cannot show that Bob has more votes than either Chuan or Diego at the point where only they remain. This newly added node will be the next selected for expansion. As node 13 has only one child, there is no difference between dive-then-expand and just expanding the node without diving. Node 14 (Alice, Bob, Chuan, Diego) will be added to the tree and evaluated. We can attack it with the assertion *NEN: Alice > Bob if only {Alice, Bob, Chuan, Diego} remain* of cost 4.5 (Figure 2.7). This cost is lower than our current lower bound on total audit cost, and so we add the assertion to our audit. In doing so, we rule out the last outcome in which Diego was a winner, and remove nodes 3 (Diego), 9 (Chuan, Diego), 13 (Bob, Chuan, Diego), and 14 (Alice, Bob, Chuan, Diego) from our tree set.

Our Audit *NEN: Chuan > Alice if only {Chuan,Alice} remain*
Chuan NEB Bob
NEN: Alice > Diego if only {Alice,Chuan,Diego} remain
NEN: Alice > Diego if only {Alice,Diego} remain
NEN: Chuan > Diego if only {Chuan,Diego,Alice} remain
NEN: Alice > Bob if only {Alice, Bob, Chuan, Diego} remain

Cost: 27

We have now ruled out all outcomes in which someone other than Chuan won. We have arrived at a slightly smaller set of assertions as we did without diving, and we created and evaluated four fewer nodes in the process! This doesn't sound like a substantial difference in effort, and it isn't for this small example. Diving is a crucial feature of RAIRE's implementation, however, for larger elections.

Interestingly, we did not end up with our redundant assertion, *NEN: Diego > Bob if only {Diego, Bob, Chuan, Alice} remain*, this time. When we didn't use diving, this assertion was added by RAIRE when it was uncertain about how much the eventual audit was going to cost. It could have removed more outcomes with the assertion *NEN: Chuan > Alice if only {Chuan,Alice} remain* at the time, but went for the cheaper option in case it could get away with a cheaper audit in the long run. Using the optimized version of RAIRE, we quickly realise our audit is going to cost at least 27 with our first dive, when expanding our first node. With this knowledge, RAIRE accepts the more expensive option to rule out the outcome (Chuan, Alice), without exploring that branch of our tree set any further.

Bibliography

Michelle Blom, Peter J Stuckey, and Vanessa Teague. RAIRE: Risk-Limiting Audits for IRV elections. *arXiv:1903.08804*, 2019.

Philip B Stark. Super-simple simultaneous single-ballot risk-limiting audits. In *EVT/WOTE*, 2010. https://www.usenix.org/legacy/events/evtwote10/tech/full_papers/Stark.pdf.

Appendix A

Specification

In this section we develop a specification for the logic of the reduction from IRV to plurality audits. The reduction consists of two separate steps:

1. a reduction from each assertion to a two-candidate contest, specifying what constitutes an over- or under-vote for each situation; and
2. a way of verifying that a set of assertions implies a particular IRV winner.

Both of these are explained in the RAIRE paper Blom *et al.* [2019], but we specify the logical details here as an aid to implementation. The reduction follows the auditing nomenclature of Stark's Super Simple Simultaneous RLAs Stark [2010].

Complete assertion scoring rules

This section describes how to implement the *Assertion Discrepancy Calculator* discussed in Part 1 of this guide.

First some definitions. A *Cast Vote Record (CVR)* is the electronic record of a ballot produced by the voting system, for example from a scanner. A *Manual Vote Record (MVR)* is a manually-entered record of a paper ballot, generally one sampled for audit.

An IRV vote may be invalid for various reasons, such as repeated, incomplete, or ambiguous preferences. Jurisdictions generally define some *interpretation rules* for such ballots, which specify how to transform an invalid vote into a valid one (generally, for IRV, an ordered list of non-repeated candidates, starting with the first preference). Raw CVRs and MVRs might be invalid votes that need to have the interpretation rules applied before counting. Most of the rest of this specification assumes that ballots are valid, *i.e.* have already had the interpretation rules applied.

Definition 1 (Interpretation rules). *The Interpretation rule function I takes a raw CVR or MVR $\tilde{b}$ and outputs a valid ballot. Notation:*

$$b = I(\tilde{b}).$$

The exact definition of I varies between jurisdictions according to relevant regulations.[1]

The following definitions are taken from Blom *et al.* [2019].

Definition 2 (IRV election). *An IRV election is defined as a tuple $\xi = (c, \mathcal{B})$ where c is the set of available candidates, and $\mathcal{B}$ a multiset*[2] *of ballots. Each ballot $b \in \mathcal{B}$ is a sequence of candidates in c, with no duplicates, listed in order of preference (most preferred to least preferred).*

We refer to sequences of candidates π in list notation (e.g., $\pi = (c_1, c_2, c_3)$), and use such sequences to represent both ballots and the order in which candidates are eliminated. We use the notation $\mathit{first}(\pi) = \pi(1)$ to denote the first candidate in a sequence π. For example, $\mathit{first}((c_1, c_2, c_3)) = c_1$.

We use the concept of *projection* to formally define a candidate's tally at any stage in the IRV counting process.

Definition 3. *Projection* $\mathbf{p}_{\mathcal{S}}(\pi)$ *We define the projection of a sequence π onto a set $\mathcal{S}$ as the largest subsequence of π that contains only elements of $\mathcal{S}$. (The elements keep their relative order in π). For example:*

$p_{\{c_2,c_3\}}([c_1, c_2, c_4, c_3]) = [c_2, c_3]$ *and* $p_{\{c_2,c_3,c_4,c_5\}}([c_6, c_4, c_7, c_2, c_1]) = [c_4, c_2]$.

This allows for a more formal version of the scoring table—see Table A.1.

The *Discrepancy* $\Delta_A(\mathit{mvr}, \mathit{cvr})$ is the extent to which cast vote record *cvr* overstates assertion A compared with actual (paper) ballot *mvr*. This generalizes the notion of discrepancy for plurality audits—rather than comparing the straightforward tallies for the winning and losing candidates according to the *mvr* and *cvr*, we compare the assertion scores for the *mvr* and *cvr* using the score function from Table A.1.

Definition 4. *For assertion A, cast vote record cvr and actual (paper) ballot mvr, the* discrepancy *is*

$$\Delta_A(\mathit{mvr}, \mathit{cvr}) = \textsc{score}_A(I(\mathit{cvr})) - \textsc{score}_A(I(\mathit{mvr})).$$

[1]For example, Colorado's rules are here: `https://www.sos.state.co.us/pubs/rule_making/CurrentRules/8CCR1505-1/Rule26.pdf`

[2]A multiset allows for the inclusion of duplicate items.

Assertion A	**Scoring vote** $\mathbf{b} = [\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}]$ SCORE$_A(b)$	**Notes**
c_1 *NEB* c_k where $k > 1$	1	Supports 1st prefs for c_1
c_j *NEB* c_k where $k > j > 1$	0	c_j precedes c_k but is not first
c_j *NEB* c_k where $k < j$	-1	a mention of c_k preceding c_j
NEN: $c_i > c_k$ if only $\{\mathcal{S}\}$ remain		
where $c_i = \mathit{first}(p_S(b))$	1	counts for c_i (expected)
where $\mathit{first}(p_S(b)) \notin \{c_i, c_k\}$	0	counts for neither c_j nor c_k
where $c_k = \mathit{first}(p_S(b))$	-1	counts for c_k (unexpected)

Table A.1: Complete scoring for vote $(c_1, c_2, \ldots, c_n)$ for all assertions. Over-statements are counted by subtracting the ballot paper score from the CVR score (and vice versa for under-statements). See Definition 4.

Assertion margins

This section describes how to implement the *Assertion Margins* component.

In order to estimate sample sizes, we need to understand the concept of "margin" as it applies to an assertion, or rather to the two-candidate contest described by that assertion.

The Assertion Margin for an assertion A is simply the sum of the scores of all ballots for A. This is the minimum number of ballots that would have to be added or removed to switch a true assertion to false.[3]

Definition 5. *The* Assertion Margin m_A *for assertion A and a set of ballots $\mathcal{B}$ is*

$$m_A(\mathcal{B}) = \sum_{b \in \mathcal{B}} \mathit{SCORE}_A(b).$$

This margin can be used directly to calculate the *diluted margin* μ for each assertion, in order to estimate the required sample size. The diluted margin for an IRV contest is the smallest diluted margin for any of its assertions.

Verifying that a set of assertions implies an IRV winner

This section describes the underlying logic of the *Assertion Visualizer and validator*. UI Design for presenting the assertions to users will be specified in consultation with UI design experts.

[3]This use of the term "margin," follows Stark's definition, i.e. the difference between the winner and the runner-up. Note that we do *not* divide the difference by 2.

A set $\mathcal{A}$ of assertions implies that w won if no other winner is possible when all the assertions in $\mathcal{A}$ are true.

Proposition 1. *A set of assertions $\mathcal{A}$ implies that w won if for all other candidates $c_n \neq w$, for all possible elimination sequences $(c_1, c_2, \ldots, c_{n-1}, c_n)$ in which c_n wins, there exist $i < k \in [1, n]$ s.t. either*

"NEN: $c_i > c_k$ if only $\{c_i, c_{i+1}, \ldots, c_n\}$ remain" $\in \mathcal{A}$

or

"c_i NEB c_k" $\in \mathcal{A}$.

Proof. By contradiction. Assume that there is some other possible winner l. Then l can win according to a specific elimination sequence $(c_1, c_2, \ldots, c_{n-1}, l)$. But then this sequence must satisfy one of the conditions above, making it impossible. □