Notes: Computational Complexity of Voting Systems in Interactive Blockchain Model

March 24, 2021

0.1 Using Off-chain Computation and Challenge-able Claimed Results to Improve Running Time of WoodSIRV

We briefly present and analyze an algorithm to compute the winner and payoffs of the proposed voting/incentive system, making use of information that is computed off-chain and included in an execute transaction.

Suppose that M voters are deciding between n alternatives, with the set of alternatives being $\{a_1, a_2, \ldots, a_n\}$.

The submitter provides their claimed quantities for the following information:

- The winner a_i
- The order in which alternatives are eliminated by IRV rounds $\{a_{j_1}, a_{j_2}, a_{j_3}, \ldots\}$
- A matrix of each voter's highest remaining alternative by round. E.g. if voter one's first choice is a_{k_1}, which is eliminated at some point after the second round to default to her second choice of a_{k_2}, and so on until her ultimate highest remaining choice in the last round is a_{k_3}, then the top row of this matrix would be

  $$[a_{k_1} a_{k_1} ... a_{k_2} ... a_{k_3}].$$

  Then there is such a row for each voter.
- A matrix of pairwise duel results between the alternatives:

$$\begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ \ldots \\ a_n \end{array} \begin{array}{c} \begin{array}{ccccc} a_1 & a_2 & a_3 & \ldots & a_n \end{array} \\ \begin{pmatrix} & 1 & 1 & \ldots & 0 \\ 0 & & 0 & \ldots & tie \\ 0 & 1 & & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & tie & 0 & \ldots & \end{pmatrix} \end{array}$$

- The margins of victory of the winner against other alternatives[1]. For example,

$$\begin{array}{c} \\ a_i \end{array} \begin{array}{c} \begin{array}{ccccc} a_1 & a_2 & a_3 & \dots & a_n \end{array} \\ \begin{pmatrix} & +3 & +2 & \dots & -1 \end{pmatrix} \end{array}$$

- The values

$$\sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j > w} \cdot w(j)$$

  for each voter.

- The lost deposits for each voter: $\{s_1, \dots s_M\}$.

- The (positive) payoff for each voter, after redistribution of fees and lost deposits: $\{\pi_1, \dots \pi_M\}$.

Note that in total this is $O(Mn \log n + n^2)$ information[2].

Then, a challenger can invalidate this submission (and receive a submitter's deposit) as follows:

- The challenger can flag an error in the pairwise duel result table, by providing the relevant pair of alternatives a_{k_1} and a_{k_2}. Then
  - The contract goes over the votes and checks how many rank a_{k_1} higher than a_{k_2}. This takes no worse than $O(Mn \log n)$ time (and with optimized data structures and if the challenger also provides the contract with the position of a_{k_1} and a_{k_2} in each vote, this might be possible in $O(M \log n)$ time).

- The challenger can flag an error in the list of eliminated alternatives by providing the index of the error, namely the round r which is reported incorrectly. Then
  - The contract creates a counter for each alternative.
  - Going over the rth column of the matrix of highest remaining votes, the contract increments the counter for each alternative each time it appears in the table.
  - The contract determines the alternative with the fewest first place votes and compares it to the claimed entry in the list of eliminated alternatives.

[1] Note we need this information only pairwise between the winner and other alternatives, so it is not necessary to include it for all pairs, for example in the preceding matrix.

[2] Note that the lost deposits, payoffs, and generally sums of the weights can be rounded to fixed precision, so the amount of information required for each entry in these lists is $O(1)$. On the other hand, expressing each entry in a list of alternatively requires $O(\log n)$ space. The margins of victory similarly are expressible in $O(\log n)$ space. The pairwise duel matrix is symmetric and could thus be compressed into $n^2/2$ entries; however, this is still $O(n^2)$.

This process requires $O(Mn \log n)$ running time; particularly one updates performs M steps updating n counters of elements expressible in $\log n$ bits.

- The challenger can flag an error in the matrix of highest remaining votes by round as follows:
  - The challenger provides both the row and the column of the error (namely the voter and the round). Let r be the number of the round/column.
  - The contract creates an internal variable: *eliminated* for each alternative.
  - The contract goes through the first $r-1$ elements of the list of eliminated alternatives and sets *eliminated* to 1 for each alternative that is present.
  - Then one checks the appropriate user's vote starting with the first ranked choice and checks whether each choice has *eliminated*$= 1$.
  - The loop stops when a choice with *eliminated*$= 0$ is found, and this is compared to the alternative claimed in the matrix.

  This process requires $O(n^2 \log n)$ time, where one takes loops of $O(n)$ steps over the number of alternatives updating n counters containing elements of $\log n$ bits.

- Them, using the information that if incorrect would have been invalidated above, the claimed winner a_i can be invalidated in one of two ways:
  - a_i is not a Condorcet winner in the round that it is claimed to be
    - The challenger provides an alternative a_j.
    - The contract verifies that $a_j \notin \{a_{j_1}, a_{j_2}, \ldots\}$ the list of eliminated alternatives.
    - The contract checks that $a_j > a_i$ in the table of pairwise duels.

    This process takes $O(n \log n)$ time.
  - Alternatively, the challenger presents an alternative a_j that should have been a Condorcet winner in a preceding round.
    - The challenger provides a_j.
    - The challenger provides the round r where a_j should have won
    - The contract creates an internal variable: *eliminated* for each alternative.
    - Then the contract goes through the first r elements of the list of eliminated alternatives, and sets *eliminated* to 1 for each alternative that is present.
    - Then the contract goes through a_j's row in the pairwise duel results table. For each a_k check if $a_k > a_j$ and if a_k has *eliminated*$= 0$.

    * If such an alternative is found, the submission is invalidated.

  This process requires $O(n^2 \log n)$ time, with the longest step being setting the *eliminated* variable for each alternative (which requires $O(n)$ steps of updating n counters containing elements expressible in $\log n$ bits).

- The challenger can flag a given margin of victory between the winner a_i and another alternative a_j as being incorrect. Then
  - The contract then goes over the votes and checks how many rank a_i higher than a_j to recompute this margin. This takes no worse than $O(Mn \log n)$ time (and with optimized data structures and if the challenger also provides the contract with the position of a_i and a_j in each vote, this might be possible in $O(M \log n)$ time).
- The challenger can flag a sum

$$\sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j > w} \cdot w(j)$$

  for a given voter as being incorrect. Then
  - Based on the margins of victory, the contract computes

$$\sum_{a_j \neq a_i} 1 - \left(\frac{|\text{margin of } a_j \text{ against } a_i|}{\text{total number of votes}} \right)^{\beta}$$

    in $O(n \log n)$ time (n steps involving objects expressible in $\log n$ bits, taking advantage of the fact that the list of margins of victory is sorted)
  - Then the contract can compute each of weights

$$w(k) = \frac{1 - \left(\frac{|\text{margin of } a_k \text{ against } a_i|}{\text{total number of votes}} \right)^{\beta}}{\sum_{a_j \neq a_i} 1 - \left(\frac{|\text{margin of } a_j \text{ against } a_i|}{\text{total number of votes}} \right)^{\beta}}$$

    in a total of $O(n \log n)$ time (n steps involving objects expressible in $\log n$ bits, taking advantage of the fact that the list of margins of victory is sorted).
  - Finally, the contract computes

$$\sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j > a_i} \cdot w(j)$$

    in $O(n \log n)$ time (n steps, involving the weights rounded to constant precision but also the alternatives which are expressible in $\log n$ bits, taking advantage of the fact that both of these lists are sorted).

Thus, this process requires $O(n \log n)$ time.

- The challenger can flag an individual lost deposit s_j as being incorrect.
  - The lost deposit is
$$D \sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j > a_i} \cdot w(j).$$
    Hence the contract can look up this value from among the list of provided sums in $O(M)$ time.
- The challenger can flag an individual payoff π_k as being incorrect.
  - The contract can compute the total amount of lost deposits using the list of lost deposits in $O(M)$ time.
  - Note that
$$\sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j < a_i} \cdot w(j)$$
$$= 1 - \sum_{a_j \neq w} \mathbf{1}_{\text{voter voted } a_j > a_i} \cdot w(j).$$
  - Then similarly to above, the contract can compute
$$\sum_{\text{voter}_k \in \mathcal{V}} \sum_{a_j \neq a_i} \mathbf{1}_{\text{voter}_k \text{ voted } a_j < a_i} \cdot w(j)$$
$$= \sum_{\text{voter}_k \in \mathcal{V}} \left(1 - \sum_{a_j \neq a_i} \mathbf{1}_{\text{voter}_k \text{ voted } a_j > a_i} \cdot w(j) \right)$$
    in $O(M)$ time.
  - Then the contract can go through the list of provided sums of weights and for each voter, in a fixed number of operations per voter, compute
$$\frac{\text{ETH fees and lost deposits}}{\sum_{\text{voter}_k \in \mathcal{V}} \sum_{a_j \neq a_i} \mathbf{1}_{\text{voter}_k \text{ voted } a_j < a_i} \cdot w(j)} \sum_{a_j \neq a_i} \mathbf{1}_{\mathcal{USR} \text{ voted } a_j < a_i} \cdot w(j).$$
    Hence this requires $O(M)$ time.

Thus, this total process requires $O(M)$ time.

Remark 1. *For the most part, one notes that the information provided by the submitter that the contract uses/trusts in the above process is already falsifiable in a previous step. So a challenger can challenge based on the first error available in the above algorithm. On the other hand, the list of eliminated alternatives and the matrix of each voter's highest remaining alternative by round are each invalidated using the other. However, this does not lead to circular reasoning as*

- *when checking the rth column of the matrix of highest remaining alternatives, one uses only the first $r-1$ entries of the list of eliminated alternatives, and*
- *when checking the rth entry of the list of eliminated alternatives, one uses only the rth column of the matrix of highest remaining alternatives.*

Hence, one can see inductively, that if there is an error in either of these pieces of information, it will be challengeable.

Remark 2. *To recap, the information that the submitter must provide is $O(Mn\log n+n^2)$ and the running time required by the contract in case of a challenge is $O(Mn\log n+n^2\log n)$. Note that, in order to be more efficient at the expense of more interactions, one could do something like the following:*

- *The submitter provides the winner and the payoffs for each juror along with a deposit (but does not provide any of the other required intermediate information above).*
- *A challenger challenges the provided information along with a deposit.*
- *Then, at this point, the submitter provides the remaining intermediate information indicated above.*
- *The challenger can highlight a specific flaw in this information, if it exists, using the above algorithm.*

Then, in cases where there are no challenges, the minimal amount of information, namely $O(M)$ is required. Only in the event of a challenge must one provide $O(Mn\log n+n^2\log n)$. However, note this approach requires a second round of interactions between submitter and challenger.

0.2 Role of Refuse to Arbitrate in Otherwise Binary Disputes

Note, from the perspective of using modules with different voting systems for different types of disputes, one could naturally use a module with the existing Plurality voting system for binary disputes. However, strictly speaking, no Kleros disputes are truly binary as Refuse to Arbitrate is always an option.

On some level, one should not allow jurors to vote Refuse to Arbitrate, *and* then also list the other alternatives as lower ranked preferences as doing so is providing arbitration services. Indeed any arbitrable application could count the highest ranking non-RtA votes as use that to determine a ruling[3]

Then, for a dispute where the alternatives are a, b, and RtA, there should only be three possible votes

[3]It might not be worth the additional code complexity to enforce this rule at the level of the contract for non-binary disputes; however, one might want to only allow RtA votes when jurors rank no other choices on the level of the UI. Regardless, if we use a module that applies Plurality voting for otherwise binary disputes, this logic is enforced naturally.

- $a > b$
- $b > a$
- RtA

Then, WoodSIRV applied to this situation simplifies to the following voting rule

- Select RtA if a majority of the voters vote for RtA
- Otherwise take whichever of a or b receives more (first-place) votes.

Similarly, the incentive system simplifies. If RtA wins, then the payoffs are equivalent to those under the current system; namely, a and b voters lose their entire deposit and these are redistributed equally among RtA voters. On the other hand, if a wins, then

- RtA voters lose their entire deposit
- a voters lose none of their deposit and receive a payoff of

$$\frac{\text{ETH fees} + \#\text{RtA voters} \cdot D}{\#\text{a voters} + w(RtA) \cdot \#\text{b voters}}$$

- b voters lose a deposit of $-w(b) \cdot D$ and receive a reward of

$$\frac{\text{ETH fees} + \#\text{RtA voters} \cdot D}{\#\text{a voters} + w(RtA) \cdot \#\text{b voters}} w(RtA).$$

If the number of RtA voters is zero, this is equivalent to the current payoff structure; as the number of RtA voters approaches half of the number of jurors, then the payoffs for the a and b voters converge.

Ultimately, whether to use the existing Plurality system or this (simplified version of) WoodSIRV for these cases comes down to how much one prioritizes the question of whether the system decides correctly whether it should arbitrate or not, compared to the question of which alternative it should choose if it does arbitrate. The simplified WoodSIRV system makes it difficult to pass an incorrect RtA result as this requires 50% support; on the other hand in particularly pathological situations this could allow a or b to win with just over 25% of the total vote.