Effective Election Tabulation

Howie Benefiel, Robert Pate, Kelsey Sandlin, Patrick Sigourney
Social Computing - Dr. Vijay Garg - Fall 2018
The University of Texas at Austin

***Abstract*— The problem of how best to tally votes in an election so as to produce a socially beneficially result has been studied for centuries with numerous resulting methodologies. For elections with Byzantine agents, the methodology must take into account bad actors among the electorate who seek to disrupt the election of the popular choice. The Kemeny-Young [1] method is a popular election method which has been modified to be fault-tolerant against Byzantine agents in what is known as the Pruned-Kemeny method [2]. Both Kemeny-Young and Pruned-Kemeny are NP-hard in that the time-complexity increases exponentially as the number of candidates in the election increases. This paper provides analysis of several election methods and proposes opportunities for improvement.**

I. INTRODUCTION

On the surface, elections appear simple enough: the population is presented with a selection of candidates, each voter chooses their favorite candidate, and the election authority tabulates the votes and declares the candidate with the most votes the winner. However cases can arise where the final results may not most accurately reflect the will of the electorate and thus not produce the greatest social welfare. This paper will investigate several popular election tabulation methods and derivations thereof. For this discussion, the methodology by which the votes are collected is not addressed; we will assume that the input to each voting method is an agreed-upon and correct ballot of ranked candidates for each participating voter.

II. RELATED WORK

A. Kemeny-Young

The Kemeny-Young method was developed by John Kemeny in 1959[1] and validated by Peyton Young as a way of reconciling the Borda and Condorcet election methods [5]. The common thought was to select a Condorcet alternative when one existed but lacking such an alternative, to fall back to the Borda method. The same results produced by these two phased requirements were achieved by a single unified method proposed by Kemeny and whose correctness was proved by Young in 1978 [5].

Kemeny prescribes a technique for tabulating votes in a ranked-choice election whereby the voters' selections are tabulated against each unique pairing of candidates similar to the Ranked Pairs method. Rather than using the pairing ranks to directly order the final results, Kemeny takes each possible permutation of candidate rankings and calculates the Kendall tau distance of the sum of voter ballots from each permutation. Kendall tau is a measure of the number of adjacent element swaps necessary to transform one ordering into another. The candidate ordering permutation which has the smallest distance from the aggregated voters' choice is the winning ordering.

Algorithm 1 Kemeny-Young

```
1:  procedure KEMENY-YOUNG
2:      P ← All permutations of k candidates
3:      B ← agreed upon pairwise ballots of votes
4:      maxScore ← 0
5:      maxRank ← null
6:      for each ranking r in P do
7:          score ← Kemeny-YoungScore(r, B)
8:          if score > maxScore then
9:              maxScore ← score
10:             maxRank ← r
        return maxRank
```

B. Pruned Kemeny

While the Kemeny-Young method works well for general cases, it does not handle situations involving Byzantine agents voting in a manner intended to reduce the social welfare of the results. In these cases, a mechanism must be applied to identify and mitigate these faulty processes. The Pruned Kemeny method was devised by Chauhan and Garg in 2013 [2] as an enhancement to Kemeny-Young which adds an additional "pruning" step to the K-Y algorithm, identifying the f ballots with the greatest Kendall tau distance from the total sum of ballots and removing these outliers from the result tally.

Algorithm 2 Pruned-Kemeny

```
1:  procedure PRUNED-KEMENY
2:      P ← All permutations of k candidates
3:      B ← agreed upon pairwise ballots of votes
4:      maxScore ← 0
5:      maxRank ← null
6:      for each ranking r in P do
7:          F ← the most distant rankings from r in B
8:          B* ← pairwise ballot after pruning F
9:          score ← Kemeny-YoungScore(r, B)
10:         if score > maxScore then
11:             maxScore ← score
12:             maxRank ← r
        return maxRank
```

C. Ranked Pairs

Ranked Pair tabulation is a method devised by Nicolaus Tideman in 1987 [7]. In it, ballot rankings are tallied against pairwise comparisons of candidates to produce a score representing voter preferences in each possible one-on-one contest. These pairing scores are used to produce an ordering of the candidates as the final election result: the list of ordered pairs are sorted in descending order by score. Starting from the highest scoring pair, the pairs are inserted one at a time into a directed graph. As the graph is constructed, any pair which would produce a cycle is discarded. The resulting graph represents the final ranking of candidates. The Ranked Pair system is a polynomial time algorithm which will elect a Condorcet candidate if one exists.

Algorithm 3 Ranked Pairs

```
1: procedure RANKED-PAIRS
2:     B ← agreed upon pairwise ballot of votes
3:     B_sorted ← B sorted by largest majority
4:     maxRank ← []
5:     for each pair p in B do
6:         if p doesn't form cycle in maxRank then
7:             maxRank.lock(p)
       return maxRank
```

D. Merge Sort

Similar to Ranked Pairs, Merge Sort begins by tallying pairwise votes to produce a score for each candidate pairing. An arbitrary ordering of candidates is produced and this ordering is sorted using merge sort where the weighting of preferences from the tally scores of each alternative pairing determines the candidate ordering. When merging, the first two elements of the `left` and `right` lists, the first two candidates in each list are compared by comparing which candidate got more votes head-to-head.

The side-effect of this is that cycles in the *is-preferred* graph are never considered and never broken. The cycles in the graph are implicitly broken by the result list being filled completely. Because Merge Sort is well-known and does not do anything sophisticated to break cycles, it is often used as a best guess for other algorithms.

III. PROBLEM STATEMENT

The problem with Kemeny-Young, and by extension Pruned-Kemeny, is its NP-hardness. The NP-hardness of Kemeny-Young stems from the requirement of finding all permutations of candidate ordering which, implemented via a recursive solution, produces a time complexity of n-factorial. While the complexity tied to the number of voters is linear [1], an election where the number of candidates is greater than three can require significant time to calculate [4]. To this end, we set about to find if a more efficient but perhaps less accurate tabulation algorithm could produce similar results to Kemeny-Young with the addition of a pruning component.

IV. PROCESS

While Kemeny-Young is generally regarded as a near-ideal method to accurately tabulate the votes of the electorate, what about cases where some voters may seek to reduce the social-benefit of an election rather than increase it? In 2013 Himanshu Chauhan and Vijay Garg proposed a variation of Kemeny-Young which accounts for these Byzantine voters. Byzantine is the term used to describe elements of a system whose choices, whether intentional or not, run contrary to the consensus of the population. Chauhan and Gargs proposal would prune out these faulty elements using the assumption that all non-faulty voters would seek to maximize social-benefit and therefore the voting patterns of good voters will be similar. Through this assumption, faulty voters can be identified as those whose voting selections differ significantly from the consensus of the group. Consequently, the votes of these elements are removed from the tally.

For this paper, we chose to investigate how the pruning of Byzantine voters could be applied to other election tabulation methods, specifically those possessing a polynomial time complexity.

A. Polynomial Time Approximation Scheme - PTAS

Our first attempt at speeding up the algorithm was to attempt to apply pruning to a known polynomial time approximation scheme for Kemeny-Young.

1) PTAS Overview: Polynomial-time approximation schemes are algorithms which attempt to estimate the optimal value, OPT(*I*), for a problem instance, *I* [6]. Let *X* be a minimization problem. Let $\epsilon > 0$ and $\rho = 1 + \epsilon$. A PTAS is called a ρ -approximation algorithm for some problem *X* if it delivers a feasible solution with objective value A(*I*) for all instances if *I* of *x* such that

$$|A(I) - OPT(I) \leq \epsilon \cdot OPT(I) \tag{1}$$

In this definition, ρ is the worst case ration of the approximation algorithm. A couple notes about this are that as $\rho \to 1$, the approximation algorithm yields a solution closer and closer to OPT(*I*).

Approximation schemes can be broken down into two classes:

- A polynomial-time approximation scheme (PTAS) has a time complexity which is polynomial in the input size of the problem instance.
- A fully polynomial time approximation scheme (FPTAS) has a time complexity which is polynomial in the input size and also polynomial in $1/\epsilon$.

The key difference here is that a PTAS can be exponential in the approximation factor. That means that a PTAS can have a time complexity proportional to $|I|^{1/\epsilon}$ where $0 < \epsilon < 1$. That is opposed to a FPTAS where the time complexity is polynomial in ϵ, so a time complexity proportional to $|I|^{10}/\epsilon^5$ would be acceptable.

Polynomial time approximation schemes are commonly developed using one of three methods. In general approximation schemes either reduce the size of the input, reduce the

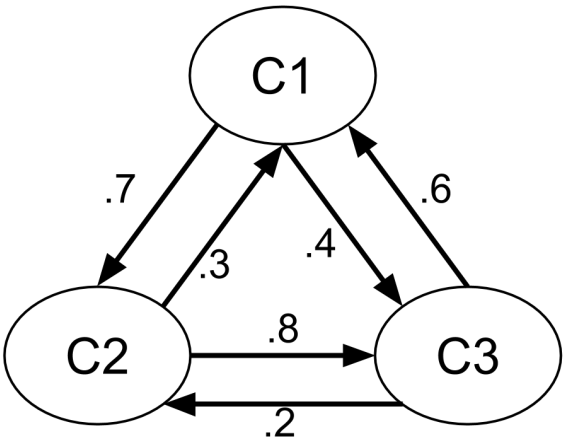


Fig. 1. Feedback arc tournament applied to democratic elections.

size of the output, or reduce the complexity of the execution of the algorithm.

2) A PTAS for Weighted Feedback Arc Set on Tournaments: The PTAS for Kemeny-Young is presented in the context of a weighted feedback arc set tournament [3]. This problem is described as the following directly from Kenyon-Mathieu and Schudy:

Problem 1: The input is a complete directed graph with vertex set V and non-negative edge weights w_{ij} with $w_{ij} + w_{ji} \in [b, 1]$ for some fixed constant $b \in (0, 1]$. The output is a total order $R(x, y)$ over V minimizing $\sum_{x,y \subset V} w_{xy} R(y, x)$.

This formulation of democratic elections is shown in Fig. 1. The goal of this algorithm is to break any cycles in the graph without causing too many upsets in the final result.

The algorithm presented in the paper is shown in Alg. 4.

Algorithm 4 PTAS for non-Byzantine Kemeny Young

procedure KEMENYYOUNGPTAS
  Input: A weighted tournament
  $\pi \leftarrow$ constant factor approximation
  while some move decreases cost **do**
    Types of moves:
    1) **Single vertex moves.** Choose vertex x and rank j, takes x out of the ordering and put it back in so that its rank is j.
    2) **Additive approximation** Choose integers $i < j$; let $U = \pi_U \leftarrow$ AddApprox(U) with parameter $\beta(\epsilon)$. Replace the restriction π_U of π to U by π'_U.
  return π

In the above algorithm, $\beta(\epsilon)$ is given by Equation 2.

$$\beta(\epsilon) = 9^{-\frac{1}{\epsilon}\log_{3/2}(1/\epsilon^2)}\epsilon^3 \tag{2}$$

And finally the time complexity of this algorithm is

$$O(n^6/\epsilon \cdot f(n, (1/\epsilon)^{O(1/\epsilon)}, 0)) \tag{3}$$

where $f(n, \beta, \eta)$ is the time required to run the additive approximation algorithm.

After seeing this PTAS algorithm, our plan was to run the pruning procedure on the result of the PTAS at each step to handle byzantine agents. After beginning to implement the PTAS, we realized that the AddApprox subroutine was troublesome. The paper pointed to another paper which described the AddApprox subroutine, but it was difficult finding which exact procedure the authors intended to be used. Then, if we could determine which procedure to use, all the procedures were extremely dense algorithms which would have taken quite some time to implement. After spending a couple weeks attempting to get the PTAS working, the group decided to apply pruning to simpler algorithms.

B. Pruning Applied to Less Sophisticated Algorithms

The pruning functionality in Pruned-Kemeny requires iterating over all permutations of the set of candidates, so a new method of pruning Byzantine voters needed to be implemented for Ranked-Pairs and Merge-Sort algorithms. The assumed goal of Byzantine voters is to minimize social welfare, so it is expected that if a majority of *good* voters prefer candidate *a* over candidate *b*, the majority of *bad* voters will attempt to oppose this pair. Using this assumption, the probability of *badness* is assigned to each voter based on the weight of each majority-preferred pair they oppose. For the Ranked-Pairs algorithm, this is accomplished by running the algorithm twice. The first round iterates over the the majority preferred pairs to find any opposing voters. The opposing voters are ranked by weight and the desired number of suspected Byzantine votes are pruned. The Ranked-Pairs algorithm is then run a second time on the pruned set of votes.

Algorithm 5 Pruned Ranked Pairs

procedure PRUNED-RANK-PAIRS
  B $\leftarrow$ agreed upon pairwise ballot of votes
  $B_{sorted} \leftarrow$ B sorted by largest majority
  voterWeights $\leftarrow$ []
  for each pair p in B **do**
    opposingVoters $\leftarrow$ any voter that opposes pair p
    for voterIndex in opposingVoters **do**
      ▷ Use the weight of the pair in the ballot
      ▷ as a probability of a voter's badness
      voterWeights[voterIndex] $\leftarrow$ B[p]
  badVoters $\leftarrow$ top f voters in voterWeights
  $B^* \leftarrow$ agreed upon ballot after pruning badVoters
  maxRank $\leftarrow$ Run Ranked-Pairs on B^*
  return maxRank

C. Comparison and Vote Simulation

We chose to replicate the simulation described in [2] in order to reproduce their results for Kemeny and Pruned Kemeny and to provide a quality comparison between them and the polynomial time pruning algorithms we were introducing.

As noted in [2], good and byzantine voters are simulated by inverting the order of candidate pairings from a given ideal such as A, B, C, D. A perfectly good voter will vote the ideal while a perfectly bad voter will vote the inverse. We also maintained the assumptions from [2] where there is only one round of voting (though it's run 50 times and averaged) and voter rankings are transmitted securely so that good voters have true rankings unaffected by byzantine voters.

This voter simulation was built into a stand alone package that holds the data in memory in groups that mirror the charts seen in this paper and provides an algorithm runner and aggregator to feed the data to multiple algorithms, calculate the average distance as described, and compile it into coordinates ready for inclusion in a report. This allows for any voting algorithm to be included in the comparison by fitting a very simple interface.

For each execution, the electorate size was fixed at n=100 total voters and f=33 Byzantine voters who will vote 'badly' with a probability of 0.9. For a Byzantine vote to be 'bad' means that for any ordered pair (a,b) within the ideal candidate ordering, there is a 90% chance that the Byzantine vote will contain an ordering of (b,a) for that pair. (Producing an ordered ranking from this randomization may not be immediately obvious so we have provided our algorithm for doing so as below prior to the results.) The probability of a non-Byzantine voter selecting ordering (a,b) for any given ordering (a,b) contained in the ideal candidate ordering is represented by the x-axis in the result graphs which follow. The higher the probability, the more likely the voter's ballot will align with the ideal ballot. The y-axis represents how far away the voters' ballots are from the ideal ranking. This value is given as the Kendall tau distance of the average of the n=100 voter ballots from the ideal.

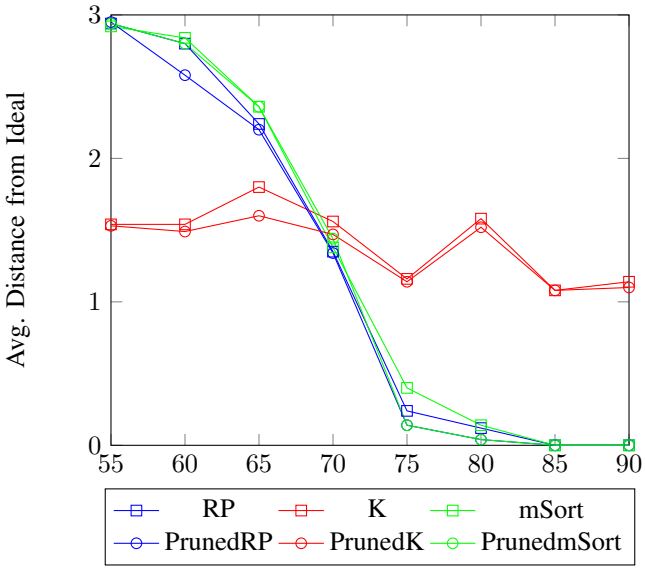
Algorithm 6 Voter Randomization

procedure RANDOMIZEVOTES(List GoalOrder, int Probability)
  Init List $voterPrefs$
  for index i in Goal Order **do**
    $c1 = GoalOrder.get(i)$
    Record $c1Position$ in $voterPref$
    If !$c1Position$ add to end (happens when i is 0)
    for index j init to $i+1$ in Goal Order **do**
      $c2$ = GoalOrder.get(j)
      Record $c2Position$ in $voterPref$
      If !$c2Position$ add after $c1Position$ (happens when i is 0)
      IsBadOrder set to $c2Position > c1Position$
      IsBadVoter set to random(0-100) >= Probability]
      (IF IsBadOrder and !IsBadVoter) OR (!IsBadOrder and IsBadVoter) THEN swap
      ▷ No else because if both bad do nothing or if both good, do nothing
  ▷ Good and Bad voters use the same algorithm, but bad voters' GoalOrder is the the ideal order inverted.

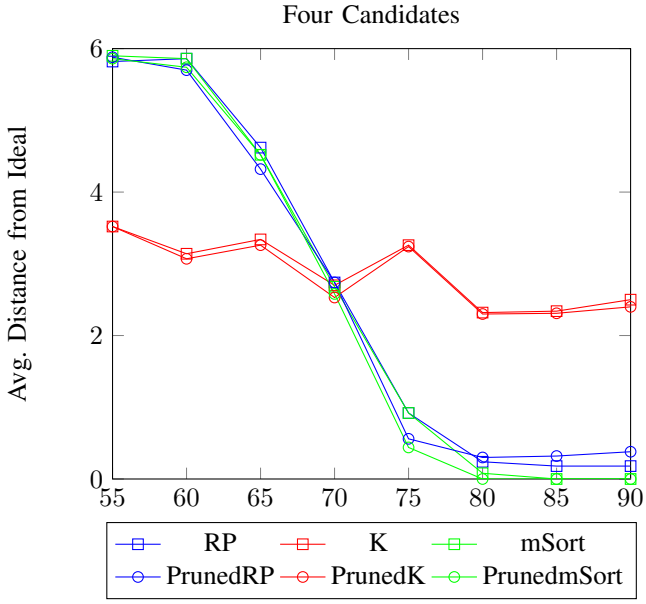
V. RESULTS

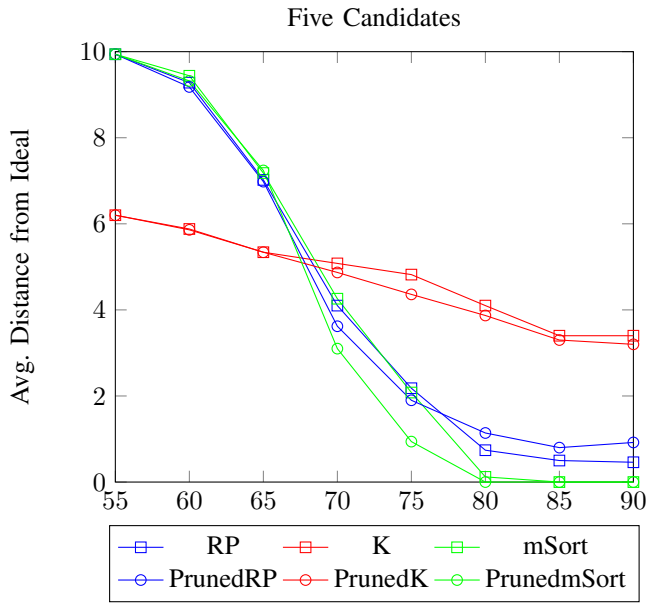
The results are shown for each algorithm for simulations of three, four, five, and six candidates. Results for seven and eight candidates were not feasible for Kemeny and Kemeny Young due to the complexity.

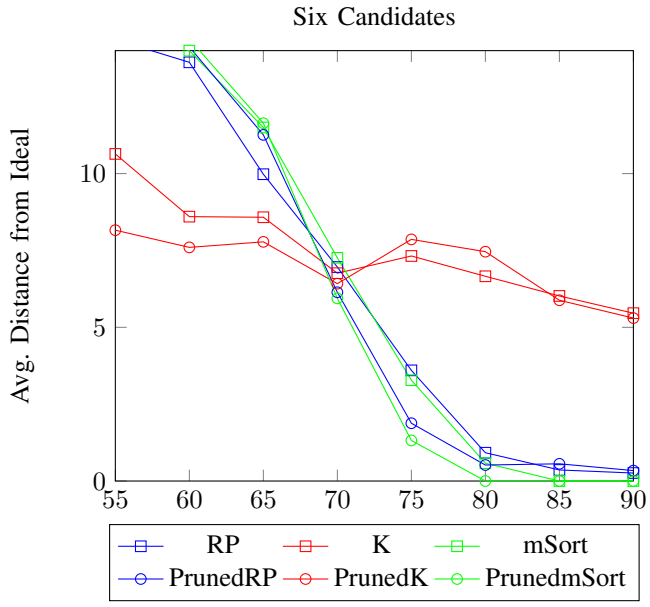
Three Candidates

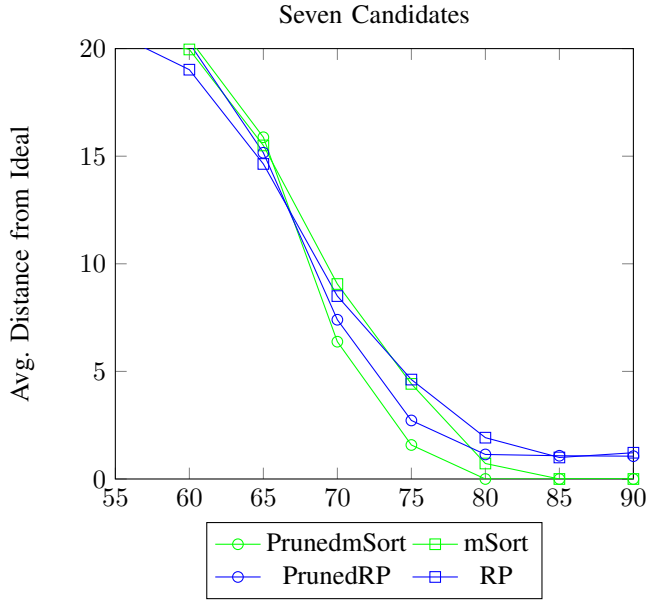


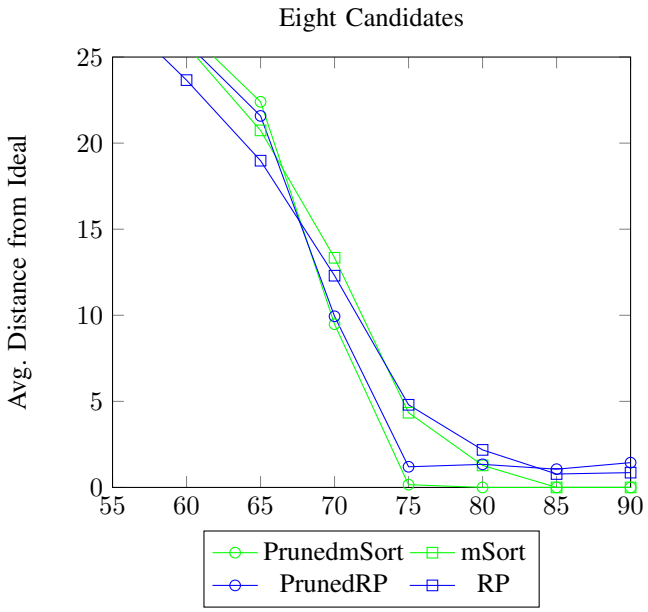
Four Candidates











VI. FINDINGS

A. *Kemeny-Young and Pruned-Kemeny Issues*

The first observation from the simulations is that the Average distance from Ideal for both Kemeny and Pruned Kemeny is much larger than observed in [2] for high probabilities of good voters. The suspected cause for this difference is an error in the implementation for the Kemeny-Young Score. For low probabilities, the results are similar. Future work includes fixing the Kemeny-Young Scoring mechanism used in the Kemeny and Pruned Kemeny implementations.

B. *Kemeny compared to Ranked Pairs and MergeSort*

When we compare Kemeny and Pruned Kemeny (NP-complete) to the other sorting algorithms (O(n(n-1)/2)), we find that Kemeny provides a result much closer to the ideal ranking. Across the simulations for three, four, five, and six candidates, the Kemeny algorithms return a ranking that is nearly a third closer to the ideal ranking. As the probability of good voter consensus increases, Ranked Pairs and MergeSort begin producing rankings much closer to the ideal. A simulation of Kemeny-Young with seven candidates took over 10 hours to complete, while a simulation of either Ranked Pairs or MergeSort completes in seconds. If the probability voter consensus is fairly high, or there is a strict bound on run time, either Ranked Pairs or MergeSort appear to be a good option.

C. *Unpruned Algorithms compared to Pruned*

In each of the simulations, Pruned-Kemeny generally does slightly better than Kemeny, Pruned Ranked Pairs does slightly better than Ranked Pairs, and Pruned MergeSort does slightly better than MergeSort. It is interesting to note that the higher the probability of consensus of voters, the more the pruned versions of Ranked Pairs and MergeSort out performs the non-pruned version. This makes intuitive sense because the pruning mechanism in each of these algorithms is weighted using the majority pairwise winners. If there is

100 percent voter consensus, there will be a majority (2/3 if 1/3 are Byzantine) for every good pair and 90 percent of the Byzantine voters will oppose that pair. As we approach 100 percent, the Byzantine voters are more detectable using majority pairwise comparisons.

VII. CONCLUSION

When we started this project, our intent was to modify the Pruned-Kemeny algorithm to achieve polynomial time complexity through approximation schemes. After many (many!) hours of research and readings, we determined a more attainable solution to address the problem is to take existing polynomial algorithms and incorporate Byzantine pruning functions into them in order to attain a polynomial time fault-tolerant election algorithm.

We found the Kemeny-Young algorithms to be more consistent in their accuracy across variations of good voter vote probability; changes in vote probability had little impact on accuracy of the Kemeny and Pruned Kemeny algorithms. This is in stark contrast to the two polynomial time algorithms which produced a high degree of accuracy when voter 'good probability' is above 75 percent but drops off quickly with less consistent voter rankings.

The pruned version of each algorithm frequently produces slightly better accuracy, but the difference is not significant. This is in contrast to the results produced by [2], we hypothesize the difference may be due to undetermined discrepancies in the test data generation methods.

REFERENCES

[1] J. Kemeny, "Mathematics without Numbers", Daedalus 88 (1959), pp. 577591.

[2] H. Chauhan, V. Garg, "Democratic Elections in Faulty Distributed Systems", The University of Texas at Austin (2013).

[3] C. Kenyon-Mathieu, W. Schudy, "How to Rank with Few Errors: A PTAS for Weighted Feedback Arc Set on Tournaments", Brown University (2007).

[4] V. Conitzer, A. Davenport, J. Kalagnanam, "Improved Bounds for Computing Kemeny Rankings" (2006), http://www.cs.cmu.edu/ conitzer/kemenyAAAI06.pdf

[5] H.P. Young, A. Levenglick, "A Consistent Extension of Condorcet's Election Principle" (1978) http://www.econ2.jhu.edu/People/Young/scans/VR16.pdf

[6] P. Schuurman, G. J. Woeginger, "Approximation Schemes - A Tutorial", Eindhoven University of Technology, http://www.win.tue.nl/ gwoegi/papers/ptas.pdf

[7] T.N. Tideman, "Independence of Clones as a Criterion for Voting Rules" (1987), Soc Choice Welfare (1987) 4: 185.