Practical Algorithms for Computing STV and Other Multi-Round Voting Rules

Chunheng Jiang
Rensselaer Polytechnic Inst.
Dept. of Computer Science
jiangc4@rpi.edu

Sujoy Sikdar
Rensselaer Polytechnic Inst.
Dept. of Computer Science
sikdas@rpi.edu

Hejun Wang
Rensselaer Polytechnic Inst.
Dept. of Computer Science
wangj38@rpi.edu

Lirong Xia
Rensselaer Polytechnic Inst.
Dept. of Computer Science
xial@cs.rpi.edu

Zhibing Zhao
Rensselaer Polytechnic Inst.
Dept. of Computer Science
zhaoz6@rpi.edu

ABSTRACT

STV is one of the most commonly-used voting rules for group decision-making, especially for political elections. However, the literature is vague about which tie-breaking mechanism should be used to eliminate alternatives. We propose the first algorithms for computing co-winners under STV, each of which corresponds to the winner under some tie-breaking mechanism. This problem is known as *parallel-universes-tiebreaking (PUT)-STV*, which is known to be NP-complete to compute [9]. We conduct experiments on synthetic data and Preflib data, and show that standard search algorithms work much better than ILP. We also explore improvements to the search algorithm with various features including pruning, reduction, caching and sampling.

1. INTRODUCTION

Voting is one of the most practical and popular ways for group decision-making, and is one of the major topics under *social choice theory*. In the past decades there has been a growing literature of *computational social choice*, which studies computational aspects of social choice problems and voting rules [4]. More recently, computational social choice, in conjunction with algorithmic game theory, has been recognized as one of the eleven "*fundamental methods and application areas*" of AI, according to The One Hundred Year Study on Artificial Intelligence [14].

One of the earliest and the most fundamental problems in computational social choice is the computation of winners of well-studied voting rules. In fact, the widely-regarded first paper in computational social choice, published by Bartholdi et al. in 1989 [3], proved that *Dodgson*'s rule and the *Kemeny* rule are NP-hard to compute. In addition, the *Slater* rule is also NP-hard to compute [7].

For political elections, the plurality rule seems to be the most common choice. Perhaps the second one is *Single Transferable Vote (STV)*, also known as *instant runoff voting, alternative vote*, or *ranked choice voting*. According to wikipedia, STV is being used to elect senators in Australia, city councils in San Francisco (CA, USA) and Cambridge (MA, USA), and has recently been approved to be used for state and federal elections in Maine State in the USA.

A typical description of STV is the following. Suppose there are m *alternatives*. In each round, we calculate the *plurality score* for each remaining alternative, which is the number of times it is ranked in the first place. The alternative with *the smallest plurality score* is eliminated. This has the effect of transferring the ballots in support of the eliminated candidate to their corresponding favorite remaining candidate. The last-standing alternative is the winner.

However, it was not clear from the literature which alternative should be eliminated when two or more alternatives are tied for the last place in a round. For example, in the San Francisco version, "*a tie between two or more candidates shall be resolved in accordance with State law*" [1]. See [2] for a list of commonly used variants of STV.

Random elimination and fixed-order tie-breaking are two popular tie-breaking mechanisms for STV. Random elimination, as the name suggests, means that whenever multiple alternatives are tied for the last place, the one to be eliminated is chosen uniformly at random. Fixed-order tie-breaking is characterized by a linear order $\mathcal{O}$, called the *priority order*, over the alternatives. Among all alternatives that are tied for the last place in a certain round, the one that is ranked lowest in $\mathcal{O}$ is eliminated. However, random elimination may result in poor ex-post satisfaction due to randomness. For fixed-order tie-breaking, it is unclear how the priority order should be determined, and the existence of such an order itself is unfair to the alternatives who are ranked low in the priority order. Formally, STV with fixed-order tie-breaking violates *neutrality*.

A natural solution is to output all alternatives who can be made to win under *some* tie-breaking mechanism. This multi-winner version of STV is called *parallel-universes-tiebreaking (PUT)-STV* [9], and the same paper proved that computing the winners under PUT-STV is NP-complete. To the best of our knowledge, no practical algorithm exists for computing PUT-STV.

NP-hardness of PUT-STV may not be a critical real issue in political elections, as the frequency of holding such elections is low, the number of alternatives is often large, and the chance of ties may not be high. The NP-hardness becomes more critical in low-stakes and more frequent group decision-making scenarios, such as a group of friends us-

Appears at: 4th Workshop on Exploring Beyond the Worst Case in Computational Social Choice (EXPLORE 2017). Held as part of the Workshops at the 16th International Conference on Autonomous Agents and Multiagent Systems. May 8th-9th, 2017. Sao Paulo, Brazil.

ing voting to decide the restaurant for dinner using an online voting website, for example Pnyx [5], robovote.org, or opra.tech. In such cases, in addition to computing all winners as soon as possible, a more practical objective is to design anytime algorithms for PUT-STV to encourage early discovery of winners for better user experience and timely decision-making.

To address this problem, we model the problem of determining the set of all co-winners under different run-off voting rules as a search problem in AI. We compare standard AI search algorithms together with various ways of improving the performance w.r.t. the following measures of performance.

- Time taken to discover all winners.
- Early discovery of a large portion of winners.

The first measure is important for high-stakes applications such as political elections, because we want to make sure that all winners are found. The second measure is important for low-stakes applications where we are given limited resources and must output as many winners as possible.

1.1 Our Contributions

We model the PUT-STV problem as a search problem and propose various algorithms with different combinations of features, including, pruning, reduction, cache, and sampling. We employ the following techniques to improve the running time of our search algorithms and to reduce the search space explored:

- **Pruning** cuts all branches that do not lead to new winners.
- **Reduction** tries to remove multiple alternatives in each round.
- **Caching** stores visited states and prevents the same states from being explored again.
- **Sampling** can be seen as a preprocessing step: we first randomly sample multiple priority orders $\mathcal{O}$ and run STV with fixed-order tie-breaking $\mathcal{O}$ to compute multiple winners to start with, before running the search algorithm.

All algorithms are tested on three types of datasets: synthetic datasets with i.i.d. rankings chosen uniformly at random, i.i.d. single-peaked rankings, and Preflib data. Our main discoveries are the following.

1. Standard search techniques from AI perform better than ILP formulations (Section 5).
2. Caching helps increase performance. Unfortunately, reductions and sampling are expensive to compute and do not provide any benefit (Section 4).
3. For single-peaked preferences, ties are rare, and the running time grows linearly with the size of the profile (Section 4.3).

We also extend our algorithms to other multi-round voting rules, including Baldwin and Coombs, which use Borda score and veto score in each round, respectively. Computing all winners under PUT-Baldwin or PUT-Coombs is NP-hard [13].

1.2 Related Work and Discussions

There is a large literature on computational complexity of winner determination under commonly-studied voting rules. In particular, computing winners of the Kemeny rule has attracted much attention from researchers in AI and theory, see for example [8, 12] and references therein. However, STV has been overlooked in the literature, despite its popularity. We are not aware of previous work on practical algorithms for PUT-STV.

In this paper we do not discuss how to choose a single winner from the output of PUT-STV, such as the president, when multiple alternatives are PUT-STV winners. This is mostly up to the decision-maker's choice. For high-stakes applications, we believe that being able to identify potential co-winners under STV w.r.t. different tie-breaking mechanisms is important in itself, because it can detect and resolve post-election dispute on tie-breaking mechanisms.

As discussed in the Introduction, we believe that the computation of PUT-STV is important not only for political elections, but also, perhaps more importantly, for everyday group decision-making scenarios. In such cases anytime algorithms are necessary, and our search algorithms naturally have anytime guarantee—they can be terminated at any time and output the winners that have been explored so far. This is another advantage of our search algorithms over ILP.

Our work is related to a recent work on computing winners of commonly-studied voting rules by MapReduce [10], where the authors proved that computing STV is P-complete. We note that STV in [10] is with a fixed-order tie-breaking mechanism, while our paper focuses on PUT-STV. Our technique can also be used to compute PUT-Ranked-Pairs, which is NP-compete to compute [6]. See [11] for more discussions on tie-breaking mechanisms in social choice.

2. PRELIMINARIES

An *election* is given by a pair $E = (\mathcal{A}, \mathcal{N})$ where $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a set of *alternatives*, and $\mathcal{N} = \{1, \ldots, n\}$ is a set of *voters*. Let $\mathcal{L}(\mathcal{A})$ denote the set of all possible linear orders on $\mathcal{A}$. A *profile* of n voters is a collection $P = (V_1, \ldots, V_n)$ of votes where for each $i \leq n, V_i \in \mathcal{L}(\mathcal{A})$. The set of all profiles on $\mathcal{A}$ is denoted by $\mathcal{P}$. A *voting rule* r is a function $r : \mathcal{P} \to \mathcal{A}$ that maps a profile to a unique *winning* alternative.

A *scoring function* is identified by a collection of scoring vectors $M = (\vec{s}_1, \ldots, \vec{s}_m)$, where for each $\hat{m}$, $\vec{s}_{\hat{m}}$ is a vector of non-negative numbers so that for every pair $k, k' \leq \hat{m}$, if $k < k'$, then $\vec{s}_{\hat{m}}(k) \geq \vec{s}_{\hat{m}}(k')$ holds. The scoring function given by M is denoted by $score_M : \mathcal{L}(\mathcal{A}) \times \mathcal{A} \to \mathbb{Z}_{\geq 0}$. For any set of alternatives $\mathcal{A}$, a linear order $V \in \mathcal{L}(\mathcal{A})$, and any alternative c ranked at position k by V, $score_M(V, c) = \vec{s}_{|\mathcal{A}|}(k)$.

We can view the scoring functions as being defined by an $m \times m$ matrix, where the rows $\hat{m} \leq m$ correspond to the scoring vector $\vec{s}_{\hat{m}}$. As an example, the *Borda* scoring function is defined by a left triangular matrix where each $\hat{m}$-th row is the vector $(\hat{m} - 1, \hat{m} - 2 \ldots, 0, \ldots, 0)$ as shown in Figure 1.

EXAMPLE 1. *Given the linear order $V = 3 \succ 1 \succ 2 \succ 4$ over the set of alternatives $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, the Borda scoring function assigns a score of 2 to alternative a_1, denoted by $score_{M_{Borda}}(V, 1) = 2$ which corresponds to the*

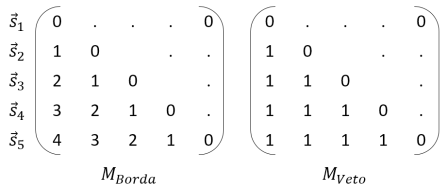


Figure 1: Matrix view of Borda and Veto scoring functions.

$(4, 2)$-th element of the matrix M_{Borda} in Figure 1.

In this paper we will consider the following voting rules.

2.1 Scoring run-off voting rules

A scoring run-off voting rule is defined by a scoring function f, and a *priority function* $g : 2^{\mathcal{N}} \to \mathcal{N}$ and proceeds in $m-1$ rounds, where at each round an alternative with the lowest score by f, ties being broken by g, is eliminated and the agents' votes are determined on the remaining alternatives. The remaining alternative is declared as the *winner*. In this paper, we are interested in the following well-studied voting rules: 1. *Single Transferable Vote (STV)* where the scoring function is the plurality function, 2. *Coomb's rule* defined by the veto function, and 3. *Baldwin's rule* defined by the Borda function.

Notice that the choice of priority function affects the outcome of the voting rule. An alternative is a *co-winner* w.r.t. a scoring run-off rule if there exists a priority function under which the alternative is declared the winner. This leads to the question: Can we determine the set of all possible winning alternatives under a given scoring run-off voting rule?

Definition 1. *(*PUT-winners*) Given a profile P, and a voting rule r, we are asked to compute the set of all co-winners.*

An alternate view of run-off voting rules is that given a profile, each voting rule corresponds to an order of eliminating the alternatives. Indeed, a brute force way to determine the set of all co-winners is to explore every possible order in which to eliminate alternatives one after the other.

This suggests two important avenues to pursue:

- Model the PUT-winners problem as a search problem, where starting at the state with all alternatives, we expand the frontier by eliminating one alternative at a time and explore the state space until we reach a state where all but one alternative remains and the corresponding alternative is a co-winner. Tracing the paths to each reachable state gives us the corresponding voting rules.
- Formulate the problem as an ILP where feasible solutions correspond to the elimination of exactly one alternative in each of $m-1$ rounds. We can test whether an alternative is a co-winner by testing whether a solution where the alternative is not eliminated in any round is feasible.

3. MODELING VOTING RULES AS A SEARCH PROBLEM

We can model the class of scoring run-off voting rules as a search problem where:

- States: there are $|2^{\mathcal{A}}| - m$ states, one for each possible elimination of 0 to $m-1$ alternatives.
  - Start state: no alternatives have been eliminated.
- Successor function: maps the current state to the set of states where an alternative with the lowest score is eliminated.
- Output: a set of winning alternatives.

Beginning from the start state, we add states to the *frontier* using the successor function. At each iteration, we choose a state from the frontier to *explore*, and remove it from the frontier. If all but one alternatives have been eliminated, add the remaining alternative to the set of winners. Otherwise, use the successor function to add new states to the frontier, one for each elimination of an alternative with the lowest score.

We use depth first search and employ the following techniques to improve the performance, and expand on them later.

(i) *pruning* involves removing a state from the frontier if all the remaining alternatives are known winners,

(ii) *reduction*, involves eliminating more than one alternative,

(iii) *caching*, involves maintaining a set of states that have been explored, and

(iv) *sampling*, where we pre-compute a subset of the possible winners by running the run-off rule using a random priority function.

Reduction Techniques A key idea in run-off voting rules is to eliminate the alternative that has the least support and run the election on the reduced problem with one less alternative. However, there are conditions under which we can remove more than one of the remaining alternatives.

For example, San Francisco STV uses the following condition [1].

If the total number of votes of the two or more candidates credited with the lowest number of votes is less than the number of votes credited to the candidate with the next highest number of votes, those candidates with the lowest number of votes shall be eliminated simultaneously and their votes transferred to the next-ranked continuing candidate on each ballot in a single counting operation.

For STV, we introduce the following generalization of the above reduction technique as follows.

Reduction for STV. In any round, suppose there is an alternative a whose plurality score is strictly larger than the total plurality score of all other alternatives with strictly less plurality scores, then those alternatives can be eliminated.

This condition guarantees that no matter what the elimination order is for the alternatives whose plurality score is strictly less than that of a, denoted by A, before alternatives in A are eliminated, none of a or $\mathcal{A} - A \cup \{a\}$ can be eliminated.

Reduction for general multi-round rules. For general multi-round rules we have a weaker reduction condition.

Given a collection of scoring vectors $M = (\vec{s}_1, \ldots, \vec{s}_m)$, and $m^* \leq m$ and any $k \leq m^* - 2$, let $\text{Diff}_M(P, m^*, k)$ denote the maximum reduction in the score difference between a pair of alternatives (a, b), before and after k alternatives have been eliminated in a ranking over m^* alternatives. $\text{Diff}_M(P, m^*, k)$ can be computed in polynomial time by enumerating all positions of a and b and all ways to eliminate k alternatives (there are no more than k^* ways, each of which corresponds to the number of eliminated alternatives that are ranked higher than a and b, between a and b, and after a and b, respectively).

The condition for general multi-round rule with scoring vectors M is: in any round, suppose there exists an alternative a with score s, let s' denote the next highest score and let A denote the alternatives whose scores are strictly less than s. If $s - s' > n \times \text{Diff}_M(P, m^*, |A|)$, then all alternatives in A can be eliminated.

It is not hard to verify the correctness of the two conditions. The condition for STV is stronger than the generic condition for computing PUT-STV.

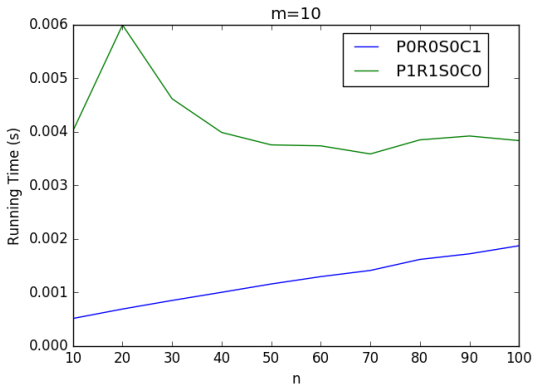


Figure 2: Comparison of runtime with and without caching on synthetic data for STV.

4. EXPERIMENTAL RESULTS: SEARCH PROBLEM

Each configuration of our experimental setup involves creating datasets of elections with m alternatives and n voters. For each dataset, we conducted experiments to evaluate the performance of depth first search while varying four parameters corresponding to whether the following techniques were used to speedup the algorithm: (i) pruning (P) (ii) reduction (R) (iii) caching (C) (iv) sampling (S) , each of which is set to 1 when the technique is used and set to 0 otherwise. Several factors affect the runtime of the search algorithm. At each iteration, we must add a branch for every alternative that is tied with the lowest score. It is easy to see how the number of ties encountered during the running of the algorithm leads to an increase in the size of the search space. In order to mitigate the effect this may have on our results, we decided to focus on the harder cases where there are ties. The profiles in each dataset are marked as (i) *easy* if there is a unique winner and every round has a unique alternative with the lowest score, and (ii) *hard* if at some round of the voting rule we encounter more than one alternative tied with the lowest score. We will focus on results for the STV rule on these hard cases.

Table 1: Summary of Preflib datasets.

	All profiles	Hard profiles
# profiles	315	49
Avg. # alternatives	25.23	77.39
Max. # alternatives	242	242
Avg. # unique orders	28.1	6.37
Max. # unique orders	4926	30
Avg. # co-winners	1.1	1.67
Max. # co-winners	4	4

Table 2: Average number of co-winners for synthetic datasets

	n									
m	10	20	30	40	50	60	70	80	90	100
10	2.89	2.04	1.88	1.78	1.72	1.64	1.57	1.58	1.53	1.53
20	4.65	4.64	4.2	3.8	3.27	3.07	2.95	2.8	2.81	2.63
30	5.58	7.24	7.4	6.95	6.51	5.85	5.84	5.33	5.05	5.06

Preflib Data In order to test the algorithms on real world preference data, we identified profiles from Preflib that have complete preferences. We found 349 profiles with complete preferences, of which 49 or about 15% correspond to hard cases (see Table 1). We find that in the real world data, profiles with ties and multiple co-winners are rare.

Synthetic Data The synthetic datasets were generated as follows: For each value of m and n, we generated profiles with n i.i.d. rankings uniformly at random over m alternatives. We then identified 1000 hard profiles to evaluate the running time and number of nodes explored to discover a given percentage of the co-winners (see Table 2).

4.1 Effect of Caching, Pruning and Reduction

Caching has the most noticeable impact on the running time (see Figure 2). Since it is a natural improvement to apply to any search problem, we leave caching on in all future experiments. The effect of pruning and applying the reduction on synthetic data is summarized in Figure 3 for the STV rule. For every configuration of $m \in \{10, 20, 30\}$ alternatives and $n \in \{10, 20, \ldots, 100\}$ voters, we generate 1000 hard profiles and report the average running time. When we apply pruning, we see a small improvement in the running time (see Figure 3(a)). However, using reductions (see Figure 3(b)) increases the average runtime and a closer inspection reveals that this was due to time spent in evaluating whether the reduction can be applied.

Our experimental results for Preflib data are summarized in Table 3. We found that for STV on real world datasets, the maximum observed running time was only 0.06 seconds.

4.2 Early Discovery and the Effect of Sampling

Our main results are focused on the more practical problem of early discovery where under a given constraint on time or computational resources, we would like to be able to discover as many of the co-winners as possible. We find that the AI search algorithms do have an early discovery property. A large percentage of the co-winners are found early in the exploration. For $m = 20$, we find that close to 80% of the co-winners are discovered after exploring just 200 states and for $m = 30$, close to half of all the co-winners are dis-

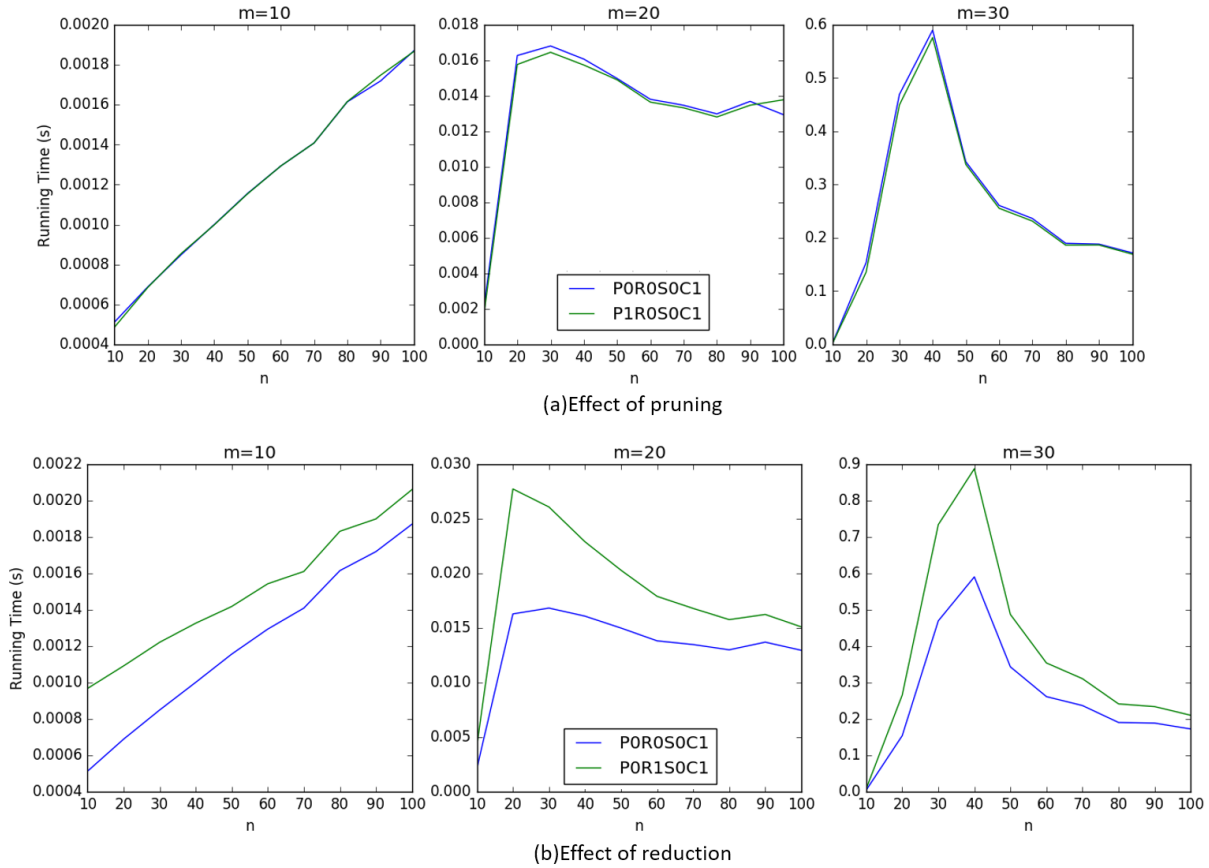


Figure 3: The effect of pruning (a) and applying the reduction (b) on the running time of the search algorithm on synthetic data for STV.

Table 3: Running time on Preflib data for STV.

Running time (10^{-4}s)	All profiles	Hard profiles
average	3.35	1.42
minimum	0.39	0.52
maximum	648.93	4.65

Table 4: Average number of ties when running the search algorithm to compute all STV co-winners.

Preferences	$m = 10$	$m = 20$	$m = 30$
Random preferences	4.7255	76.574	1324.301
Single-peaked preferences	1.9354	3.7466	6.4802

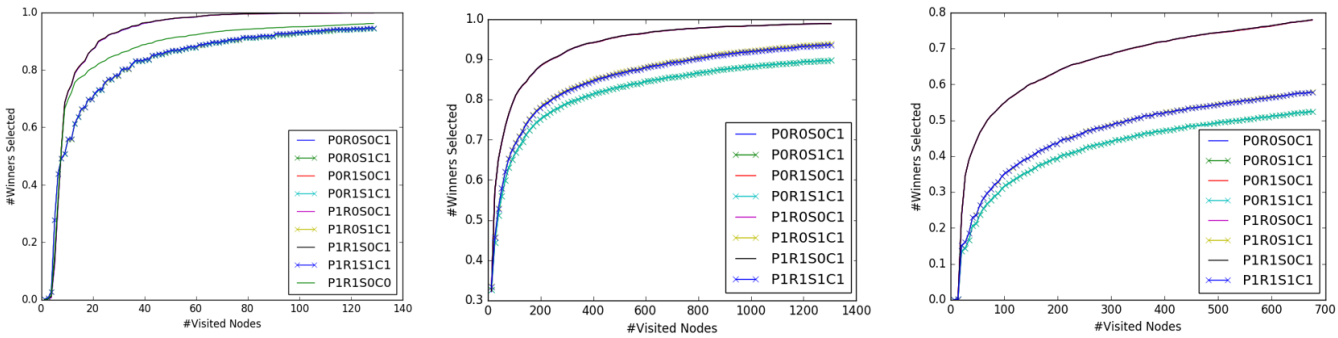
covered after exploring only 100 states. Somewhat unsurprisingly even a relatively unsophisticated search strategy without any improvements to reduce the search space other than caching performs significantly better than attempting to discover co-winners by breaking ties at random.

For comparison, we include the running time of the search algorithms for Coomb's rule and Baldwin's rule in Figure 5. Intuitively, we expect to see a lot more ties when computing co-winners for Coomb's rule and fewer ties for Baldwin's rule and we can observe its effect on the running times which are larger than STV in general for Coomb's rule and significantly lower for Baldwin's rule.

4.3 AI Search for Single Peaked Profiles

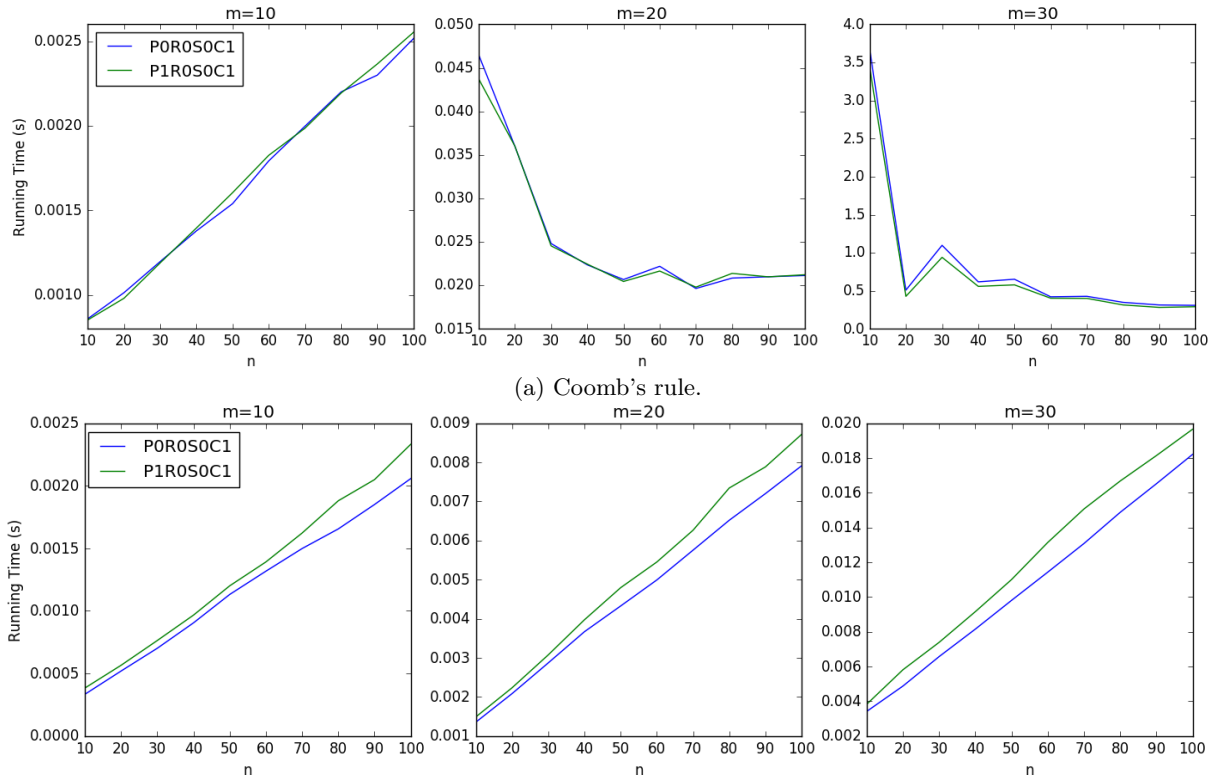
We generated profiles with i.i.d. single-peaked preferences by following the algorithm developed in [15]. For each configuration of m alternatives and n candidates, we identified 500 hard profiles from a set of randomly generated profiles. Most of the profiles have either a unique winner or only few co-winners (Figure 6(a)) and the median candidate (who is the Condorcet winner) is most frequently the co-winner of the election under the STV rule (Figure 6(b)).

We find that for a given number of candidates, the average running time of the search algorithm to compute all co-winners for single peaked profiles is significantly faster than the average time for profiles with random preferences (Figure 7) and only grows linearly with the size of the profile. For example, with $m = 30$ alternatives and profiles with 100 voters, the running time of the algorithm to compute all co-winners is under 0.003 seconds on average when preferences are single peaked which is 2 orders of magnitude lower than the average runtime for random preferences which is close to 0.2 seconds. Indeed, while the running time only increased almost linearly with the number of voters for single-peaked preferences, we observe a near exponential increase in running time with the number of candidates in the election when profiles have random preferences. This is not surprising when we consider the number of ties encountered



(a) m = 10, n = 10 (b) m = 20, n = 20 (c) m = 30, n = 30

Figure 4: Early discovery of co-winners on synthetic data for STV.



(a) Coomb's rule.

(b) Baldwin's rule.

Figure 5: Running time of the search algorithm on synthetic data with and without pruning for the Coomb's and Baldwin's rules.

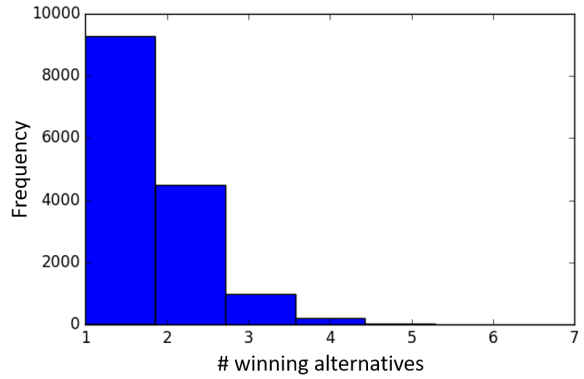
on average by the search algorithm as shown in Table 4.

5. ILP FORMULATION

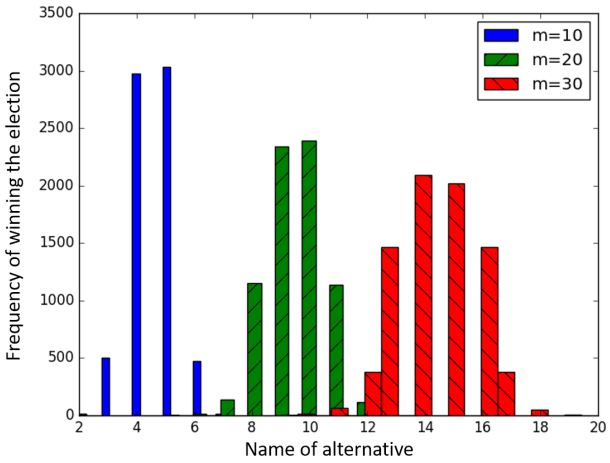
We model the PUT-WINNERS problem as an ILP where the solutions correspond to the elimination of a single alternative in each of $m-1$ rounds and we test whether a given alternative is the co-winner by checking if there is a feasible solution when we enforce the constraint that the given alternative is not eliminated in any of the rounds. We present ILP formulations of the STV and Baldwin's voting rules below. The ILP for Coomb's rule is similar to the ILP for STV where the scoring rule is changed from plurality to veto. For each alternative $a_i \in \mathcal{A}$, and for each round $t \leq m-1$, we define the variable $x_i^t \in \{0,1\}$ to model the elimination of a_i at round t.

Table 5: ILP for STV rule.

m	n	Profiles	ILP alternatives	Uncertain alternatives	Runtime(s)
10	10	392	6.51	3.52	13.026
10	20	104	8.71	6.91	961.64
10	30	88	9.10	7.49	1799.82
20	10	224	7.93	3.34	177.31
20	20	4	12.25	12	4438.66



(a) Distribution of the # winners.



(b) Frequency of alternative being the winner.

Figure 6: Single-peaked profiles generated i.i.d.

Table 6: ILP for Coomb's Rule.

m	n	# profiles	% Uncertain candidates	Runtime (s)
10	10	10	83%	2289.39
10	20	5	92%	2503.07

5.1 STV and Coomb's rule

- For alternative a_i and rounds $t \leq m$, there is a binary variable x_i^t that represents the elimination order. $x_i^t = 1$ if and only if a_i is eliminated in t-th round.
- For each $i \leq m$, $1 \leq t \leq m-1$ and $j \leq n$, there is a binary variable $p_{i,j}^t$ that represents alternative a_i's plurality score in vote V_j in round t.

The constraints are

- The usual constraints for x_i^t to be a full ranking.
- The constraint for $p_{i,j}^t$ of alternative a_i at the top position: $p_{ij}^1 = 1$ and $\sum_{t'=1}^{t-1} x_i^{t'} + p_{ij}^t = 1$
- The constraint for $p_{i,j}^t$ for alternatives from the second position: Let $K_i^t = \frac{\sum_{i' \succ_j i} \sum_{t'=1}^{t} x_{i'}^{t'}}{|\{i' \succ_j i\}|}$, then the constraint is,

$$K_i^t - \frac{m-1}{m} \leq p_{i,j}^t \leq K_i^t$$

.

- For each t, let $Plu_i^t = \sum_j p_{i,j}^t$. Then for all different i, i', we have

$$(1 + \sum_{t' \leq t} x_{i'}^{t'} - x_i^t) \times M + Plu_{i'}^t \geq Plu_i^t$$

We determine the set of all co-winners as follows: Pick an alternative a_i. Add constraints $\forall t \leq m-1, x_i^t = 0$. If ILP is feasible, a_i is a co-winner.

5.2 Baldwin's rule

The variables are: for all $i, t \leq m$, there is a binary variable x_i^t that represents the elimination order. $x_i^t = 1$ if and only if a_i is eliminated in t-th round.

The constraints are

- The usual constraints for x_i^t to be a full ranking.
- For each t, let $Plu_i^t = \sum_j (1 + |\{i' \prec i\}| - \sum_{i' \prec i} \sum_{t'=1}^{t-1} x_{i'}^{t'})$. Then for all different i, i', we have

$$(1 + \sum_{t' \leq t} x_{i'}^{t'} - x_i^t) \times M + Plu_{i'}^t \geq Plu_i^t$$

The set of co-winners is computed as follows: Pick an alternative a_i. Add constraints $\forall t \leq m-1, x_i^t = 0$. If ILP is feasible, a_i is a co-winner.

5.3 Comparison of ILP to AI Search

Tables 5, 6 and Figure 8 summarize the experimental results from running the ILPs to solve PUT-WINNERS w.r.t. STV, Coomb's rule and Baldwin's rule respectively. The results were obtained using Matlab's ILP solver. It is clear that the ILP solver takes far longer to solve PUT-WINNERS than even the default formulation of AI search without applying any of our speed up techniques. Another major problem we encountered was that the Matlab's ILP solver frequently terminates without being able to determine if the problem is feasible. A simple comparison with the results in Figure 3 reveals that the running times of standard search algorithms from AI are orders of magnitude lower than the running times of the ILP solvers.

6. SUMMARY AND FUTURE WORK

We have made the first steps of designing practical algorithms for computing all winners under STV and other multi-stage rules. We have shown that standard search algorithms are much faster and more reliable than ILP. By running experiments on synthetic dataset, we observe that cache is the most effective feature for improving running time. The algorithms run much faster on i.i.d. generated single-peaked preferences and the winners are around the median. For Preflib data, about 15% profiles we tested need tie-breaking under STV.

There are many more strategies we plan to explore. Suppose we use priority queue to store and sort the nodes to be explored, what is a good priority function to encourage early discovery of new winners? We tried multiple priority functions, such as various weighted combinations of the

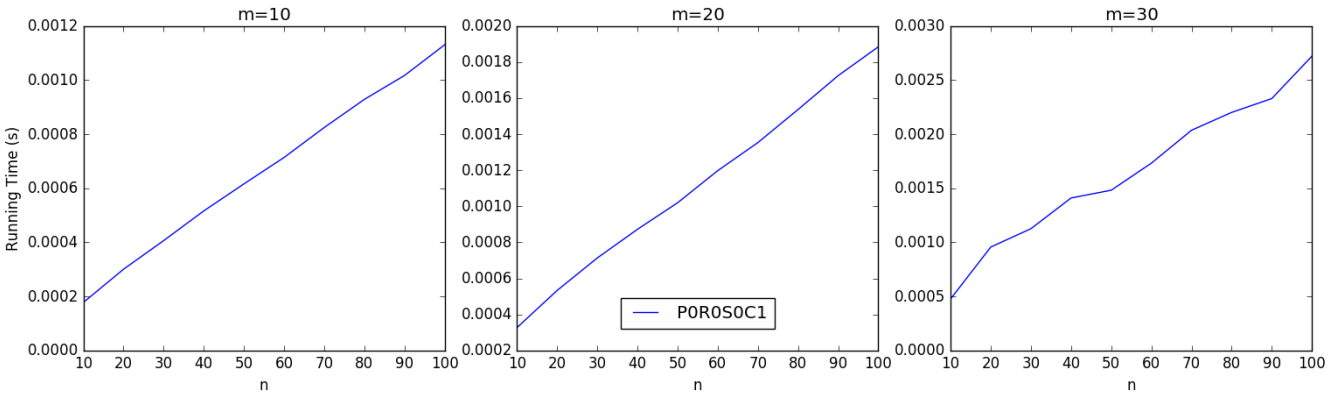


Figure 7: Running time of AI search algorithms on single-peaked preferences.

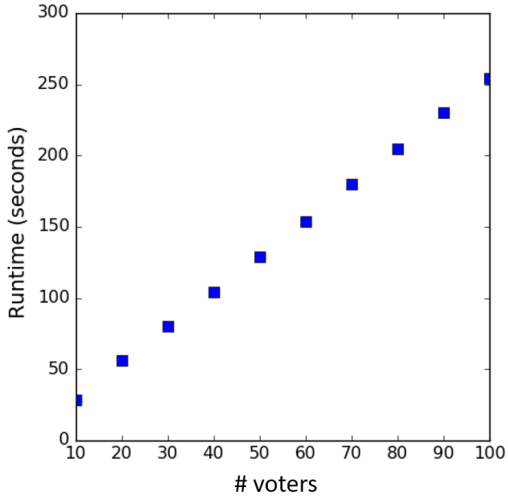


Figure 8: Average runtime of the ILP for Baldwin's rule on elections with 10 candidates as we vary the number of voters.

depth of the node, the new winners in the set, and other features. Unfortunately none of them is significantly better than the standard search algorithm. The next step is to use machine learning to learn a good priority function. A related approach is to use heuristics along with a search algorithm such as A* search. We are interested in designing good heuristics to inform the search algorithms.

A major challenge in practice, is the limited cache size. When $m = 60$ our machine sometimes run out of memory. Even if the size of memory is not an issue, the time for checking whether a node has been visited will become significant when m becomes large. How to design a good search algorithm with limited memory is an interesting and important open question. In addition to multi-round rules, we also plan to extend our techniques to compute PUT-ranked-pairs. We will integrate our algorithms for STV and other multi-stage rules to OPRA for everyday group decision-making tasks.

Acknowledgments

We thank Vincent Conitzer, Dominik Peters, Toby Walsh, for helpful discussions and suggestions.

REFERENCES

[1] `https://www.opavote.com/methods/instant-runoff-voting#san-francisco-rcv`.

[2] `https://www.opavote.com/methods/single-transferable-vote`.

[3] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[4] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[5] F. Brandt and G. C. C. Geist. Pnyx:: A Powerful and User-friendly Tool for Preference Aggregation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1915–1916, 2015.

[6] M. Brill and F. Fischer. The Price of Neutrality for the Ranked Pairs Method. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1299–1305, Toronto, Canada, 2012.

[7] V. Conitzer. Computing Slater rankings using similarities among candidates. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 613–619, Boston, MA, USA, 2006. Early version appeared as IBM RC 23748, 2005.

[8] V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 620–626, Boston, MA, USA, 2006.

[9] V. Conitzer, M. Rognlie, and L. Xia. Preference functions that score rankings and maximum likelihood estimation. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 109–115, Pasadena, CA, USA, 2009.

[10] T. Csar, M. Lackner, R. Pichler, and E. Sallinger. Winner Determination in Huge Elections with MapReduce. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[11] R. Freeman, M. Brill, and V. Conitzer. General Tiebreaking Schemes for Computational Social Choice. In *Proceedings of the 2015 International*

Conference on Autonomous Agents and Multiagent Systems, pages 1401–1409, 2015.

[12] C. Kenyon-Mathieu and W. Schudy. How to Rank with Few Errors: A PTAS for Weighted Feedback Arc Set on Tournaments. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, pages 95–103, San Diego, California, USA, 2007.

[13] N. Mattei, N. Narodytska, and T. Walsh. How hard is it to control an election by breaking ties? In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 1067–1068, 2014.

[14] P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, K. Leyton-Brown, D. Parkes, W. Press, A. Saxenian, J. Shah, M. Tambe, and A. Teller. Artificial intelligence and life in 2030. One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel, Stanford University, Stanford, CA, September 2016.

[15] T. Walsh. Generating single peaked votes. *arXiv preprint arXiv:1503.02766*, 2015.