Algorithmic Techniques for Necessary and Possible Winners

VISHAL CHAKRABORTY, University of California Santa Cruz, USA
THEO DELEMAZURE, Ecole Normale Superieure, France
BENNY KIMELFELD, Technion - Israel Institute of Technology, Israel
PHOKION G. KOLAITIS, University of California Santa Cruz and IBM Research, USA
KUNAL RELIA, New York University, USA
JULIA STOYANOVICH, New York University, USA

We investigate the practical aspects of computing the necessary and possible winners in elections over incomplete voter preferences. In the case of the necessary winners, we show how to implement and accelerate the polynomial-time algorithm of Xia and Conitzer. In the case of the possible winners, where the problem is NP-hard, we give a natural reduction to Integer Linear Programming (ILP) for all positional scoring rules and implement it in a leading commercial optimization solver. Further, we devise optimization techniques to minimize the number of ILP executions and, oftentimes, avoid them altogether. We conduct a thorough experimental study that includes the construction of a rich benchmark of election data based on real and synthetic data. Our findings suggest that, the worst-case intractability of the possible winners notwithstanding, the algorithmic techniques presented here scale well and can be used to compute the possible winners in realistic scenarios.

CCS Concepts: • **Theory of computation** → *Algorithm design techniques*; **Theory and algorithms for application domains**; • **Mathematics of computing** → *Solvers;*

Additional Key Words and Phrases: Computational social choice, elections, partial preferences, integer linear programming, poset generation methods

ACM Reference format:
Vishal Chakraborty, Theo Delemazure, Benny Kimelfeld, Phokion G. Kolaitis, Kunal Relia, and Julia Stoyanovich. 2021. Algorithmic Techniques for Necessary and Possible Winners. *ACM/IMS Trans. Data Sci.* 2, 3, Article 22 (July 2021), 23 pages.
https://doi.org/10.1145/3458472

This work was supported in part by National Science Foundation (NSF) Grants No. 1814152 and 1916647, and by US-Israel Binational Science Foundation (BSF) 201775.
Authors' addresses: V. Chakraborty, Computer Science and Engineering Department, Jack Baskin School of Engineering, University of California Santa Cruz, Santa Cruz, CA 95064, USA; email: vchakrab@ucsc.edu; T. Delemazure, Ecole Normale Superieure, 45, Rue d'Ulm, 75005 PARIS, France; email: theo.delemazure@ens.fr; B. Kimelfeld, The Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa 3200003, Israel; email: bennyk@cs.technion.ac.il; P. G. Kolaitis, Computer Science and Engineering Department, Jack Baskin School of Engineering, University of California Santa Cruz, Santa Cruz, CA 95064, USA; email: kolaitis@ucsc.edu; K. Relia, Computer Science and Engineering Department, Tandon School of Engineering, New York University, 370 Jay Street, Brooklyn, NY 11201, USA; email: krelia@nyu.edu; J. Stoyanovich, Center for Data Science, New York University, 60 5th Avenue, New York, NY 10011, USA; email: stoyanovich@nyu.edu.

2577-3224/2021/07-ART22 $15.00
https://doi.org/10.1145/3458472

1 INTRODUCTION

The theory of social choice focuses on the question of how preferences of individuals can be aggregated in such a way that the society arrives at a collective decision. It has been of interest throughout the history of humankind, from the analysis of election manipulation by Pliny the Younger in Ancient Rome to the 18th-century studies of voting rules by Jean-Charles de Borda and Marquis de Condorcet, and the more recent ground-breaking work on dictatorial vote aggregation by Kenneth Arrow in the 1950s. Over the past two decades, *computational social choice* has been developing as an interdisciplinary area between social choice theory, economics, and computer science, where the central topics of study are the computational and algorithmic perspectives of voting challenges such as vote aggregation [6].

A voting rule determines how the collection of voter preferences over a set of candidates is mapped to the set of winning candidates (the winners). Among the most extensively studied is the class of *positional scoring rules*, where every candidate receives a score from every voter that is determined only by the position of the candidate in the voter's ranking. A candidate wins if he or she achieves the highest total score—the sum of scores it receives from each voter.

A particularly challenging computational aspect arises in situations in which voter preferences are only *partial* (i.e., they can be modeled as partial orders). This might happen since, for example, voters may be undecided about some candidates or, simply, only partial knowledge of the voter's preference is available (e.g., knowledge is inferred indirectly from opinions on issues). The problem already manifests itself at the semantic level: what is the meaning of vote aggregation in the presence of incompleteness, if voting rules require complete knowledge? For this reason, Konczak and Lang [15] introduced the notions of *necessary winners* and *possible winners* as the candidates who win in *every* completion, and, respectively, *at least one* completion of the given partial preferences.

This work led to a classification of the computational complexity of the necessary and possible winners for a large variety of voting rules [2, 3, 20]. Specifically, under (efficiently computable) positional scoring rules, the necessary winners can be computed in polynomial time via the algorithm of Xia and Conitzer [20]. The possible winners can be computed in polynomial time under the plurality and veto rules, but their computation is NP-hard for every other *pure* rule, as established in a sequence of studies [2, 3, 15, 20]. Here, *pure* means that the scoring vector for m candidates is obtained from that for $m - 1$ candidates by inserting a new score into the vector.

In this article, we explore the practical aspects of computing the necessary and possible winners. Specifically, we investigate the empirical feasibility of this challenge, develop algorithmic techniques to accelerate and scale the execution, and conduct a thorough experimental evaluation of our techniques. For the necessary winners, we show how to accelerate the Xia and Conitzer algorithm through mechanisms of early pruning and early termination. For the possible winners, we focus on positional scoring rules for which the problem is NP-hard. We first give a natural polynomial-time reduction of the possible winners to **Integer Linear Programming (ILP)** for all positional scoring rules. Note that ILP has been used in earlier research on the complexity of voting problems as a theoretical technique for proving upper bounds (fixed-parameter tractability) in the parameterized complexity of the possible winners [4, 14, 21] or in election manipulation problems involving complete preferences [17]. Here, we investigate the use of ILP solvers to compute the possible winners in practice. Our experiments on a leading commercial ILP solver (Gurobi v8.1.1) show that the reduction produces ILP programs that are often too large to load and too slow to solve. For this reason, we develop several techniques to minimize or often eliminate ILP computations and, when the use of ILP is unavoidable, to considerably reduce the size of the ILP programs.

We conduct an extensive experimental study that includes the construction of a rich benchmark of election data based on both real and synthetic data. Our experimental findings suggest that,

the worst-case intractability of the possible winners notwithstanding, the algorithmic techniques presented here scale well and can be used to compute the possible winners in realistic scenarios. An important contribution of our work that is of independent interest is a novel generative model for partially ordered sets, called the **Repeated Selection Model (RSM)**. We believe that RSM may find uses in other experimental studies in computational social choice.

2 PRELIMINARIES AND EARLIER WORK

Voting profiles. Let $C = \{c_1, c_2, c_3, \ldots, c_m\}$ be a set of *candidates* and let $V = \{v_1, \ldots, v_n\}$ be a set of voters. A *(complete) voting profile* is a tuple $\mathbf{T} = (T_1, \ldots, T_n)$ of total orders of C, where each T_l represents the ranking (preference) of voter v_l on the candidates in C. Formally, each T_l is a binary relation $\succ_{T_l}$ on C that is irreflexive (i.e., $c_i \not\succ_{T_l} c_i$, for all i), antisymmetric (i.e., $c_i \succ_{T_l} c_j$ implies $c_j \not\succ_{T_l} c_i$, for all $i \neq j$), transitive (i.e., $c_i \succ_{T_l} c_j$ and $c_j \succ_{T_l} c_k$ imply $c_i \succ_{T_l} c_k$, for all i, j, k), and total (i.e., $c_i \succ_{T_l} c_j$ or $c_j \succ_{T_l} c_i$ holds for all $i \neq j$). Similarly, a *partial voting profile* is a tuple $\mathbf{P} = (P_1, \ldots, P_n)$ of partial orders on C, where each P_l represents the partial preferences of voter v_l on the candidates in C; formally, each P_l is a binary relation on C that is irreflexive, antisymmetric, and transitive (but not necessarily total). A *completion* of a partial voting profile $\mathbf{P} = (P_1, \ldots, P_n)$ is a complete voting profile $\mathbf{T} = (T_1, \ldots, T_n)$ such that each T_l is a completion of the partial order P_l; that is to say, T_l is a total order that extends P_l. Note that, in general, a partial voting profile may have exponentially many completions.

Voting rules. We focus on *positional scoring rules*, a widely studied class of voting rules. A positional scoring rule r on a set of m candidates is specified by a scoring vector $\mathbf{s} = (s_1, \ldots, s_m)$ of non-negative integers, called the *score values*, such that $s_1 \geq s_2 \geq \cdots \geq s_m$. Suppose that $\mathbf{T} = (T_1, \ldots, T_n)$ is a total voting profile. The score $s(T_l, c)$ of a candidate c on T_l is the value s_k, where k is the position of candidate c in T_l. The *score* of c under the positional scoring rule r on the total profile $\mathbf{T}$ is the sum $\sum_{l=1}^{n} s(T_l, c)$. A candidate c is a *winner* if c's score is greater than or equal to the scores of all other candidates; similarly, c is a *unique winner* if c's score is greater than the scores of all other candidates. The set of all winners is denoted by $\mathrm{W}(r, \mathbf{T})$.

We consider positional scoring rules that are defined for every number m of candidates. Thus, a *positional scoring rule* is an infinite sequence $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m, \ldots$ of scoring vectors such that each $\mathbf{s}_m$ is a scoring vector of length m. Alternatively, a positional scoring rule is a function r that takes as argument a pair (m, s) of positive integers with $s \leq m$ and returns as value a non-negative integer $r(m, s)$ such that $r(m, 1) \geq r(m, 2) \ldots \geq r(m, m)$. We assume that the function r is computable in time polynomial in m, and hence the winners can be computed in polynomial time. Such a rule is *pure* if the scoring vector $\mathbf{s}_{m+1}$ of length $(m + 1)$ is obtained from the scoring vector $\mathbf{s}_m$ of length m by inserting a score in some position of $\mathbf{s}_m$, provided that the decreasing order of score values is maintained.

As examples, the *plurality* rule is given by scoring vectors of the form $(1, 0, \ldots, 0)$, while the *veto* rule is given by scoring vectors of the form $(1, 1, \ldots, 1, 0)$. The plurality rule is the special case $t = 1$ of the *t-approval* rule, in which the scoring vectors start with t ones and then are followed by zeros. In particular, the 2-*approval* rule has scoring vectors of the form $(1, 1, 0, \ldots, 0)$. The *Borda* rule, also known as the *Borda count*, is given by scoring vectors of the form $(m - 1, m - 2, \ldots, 0)$.

Necessary and possible winners. Let r be a voting rule and $\mathbf{P}$ a partial voting profile.

- The set $\mathrm{NW}(r, \mathbf{P})$ of the *necessary winners* with respect to r and $\mathbf{P}$ is the intersection of the sets $\mathrm{W}(r, \mathbf{T})$, where $\mathbf{T}$ varies over all completions of $\mathbf{P}$. Thus, a candidate c is a *necessary winner* with respect to r and P, if c is a winner in $\mathrm{W}(r, \mathbf{T})$ for every completion $\mathbf{T}$ of $\mathbf{P}$.

- The set $\mathrm{PW}(r, \mathbf{P})$ of the *possible winners* with respect to r and $\mathbf{P}$ is the union of the sets $\mathrm{W}(r, \mathbf{T})$, where $\mathbf{T}$ varies over all completions of $\mathbf{P}$. Thus, a candidate c is a *possible winner* with respect to r and $\mathbf{P}$, if c is a winner in $\mathrm{W}(r, \mathbf{T})$ for at least one completion $\mathbf{T}$ of $\mathbf{P}$.

The notions of *necessary unique winners* and *possible unique winners* are defined in an analogous manner. The preceding notions were introduced by Konczak and Lang [15]. Through a sequence of subsequent investigations by Xia and Conitzer [20], Betzler and Dorn [3], and Baumeister and Rothe [2], the following classification of the complexity of the necessary and the possible winners for *all* pure positional scoring rules was established.

Theorem 2.1 (Classification Theorem). *The following statements hold:*

- *If r is a pure positional scoring rule, there is a polynomial-time algorithm that, given a partial voting profile $\mathbf{P}$, returns the set $\mathrm{NW}(r, \mathbf{P})$ of necessary winners.*
- *If r is the plurality rule or the veto rule, there is a polynomial-time algorithm that, given a partial voting profile $\mathbf{P}$, returns the set $\mathrm{PW}(r, \mathbf{P})$ of possible winners. For all other pure positional scoring rules, the following problem is NP-complete: given a partial voting profile $\mathbf{P}$ and a candidate c, is c a possible winner w.r.t. r and $\mathbf{P}$?*

Furthermore, the same classification holds for necessary unique winners and possible unique winners.

In the preceding theorem, the input partial voting profiles consist of arbitrary partial orders. There has been a growing body of work concerning the complexity of the possible winners when the partial voting profiles are restricted to special types of partial orders. The main motivation for pursuing this line of investigation is to determine whether or not the complexity of the possible winners drops from NP-complete to polynomial time w.r.t. some scoring rules (other than plurality and veto), if the input voting profiles consist of restricted partial orders that also arise naturally in real-life settings. We now describe two types of restricted partial orders and state relevant results.

Definition 2.2. Let $\succ$ be a partial order on a set C.

- We say that $\succ$ is a *partitioned preference* if C can be partitioned into disjoint subsets $A_1, \ldots, A_q$ so that the following hold:
  (a) for all $i < j \leq q$, if $c \in A_i$ and $c' \in A_j$, then $c \succ c'$;
  (b) for each $i \leq q$, the elements in A_i are incomparable under $\succ$; that is to say, $a \not\succ b$ and $b \not\succ a$ hold, for all $a, b \in A_i$.
- We say that $\succ$ is a *partial chain* if it consists of a linear order on a non-empty subset C' of C.

Partitioned preferences relax the notion of a total order by requiring that there is a total order between sets of incomparable elements. As pointed out by Kenig [12], partitioned preferences "were shown to be common in many real-life datasets, and have been used for learning statistical models on full and partial rankings." Furthermore, partitioned preferences contain ***doubly truncated ballots* (DTBs)** as a special case, where there is a complete ranking of top elements and a complete ranking of bottom elements, and all remaining elements between the top and the bottom elements are pairwise incomparable. This models, for example, the setting in which a voter has complete rankings of some top candidates and of some bottom candidates but is indifferent about the remaining candidates in the "middle." DTBs contain ***top-truncated ballots* (TTBs)** and ***bottom-truncated ballots* (BTBs)** as special cases. As the names suggest, in a top-truncated ballot (respectively, bottom-truncated ballot), a complete ranking of top (respectively, bottom) elements is provided, while all remaining elements are pairwise incomparable.

Partial chains arise in settings where there is a large number of candidates but a voter has knowledge of only a subset of them. For example, a voter may have a complete ranking of movies that the voter has seen but, of course, does not know how to compare these movies with movies

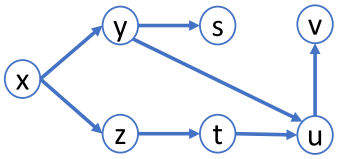


Fig. 1. A partial order, where $x \rightarrow y$ denotes $x \succ_P y$.

that the voter has not seen. Partial chains also model the setting of an election in which one or more candidates enter the race late, and so a voter has a complete ranking of the original candidates but does not know yet how to rank the new candidates who entered the race late.

Let r be a pure positional scoring rule. We write PW-PP(r) to denote the restriction of the possible winners problem w.r.t. r to partial voting profiles consisting of partitioned preferences. More formally, PW-PP(r) is the following decision problem: given a partial voting profile $\mathbf{P}$ consisting of partitioned preferences and a candidate c, is c a possible winner w.r.t. r and $\mathbf{P}$? Similarly, we write PW-PC(r) to denote for the restriction of the possible winners problem w.r.t. r to partial voting profiles consisting of partial chains.

Kenig [12] established a nearly complete classification of the complexity of the PW-PP(r) problem for pure positional scoring rules. In particular, if r is the 2-approval rule, then PW-PP(r) is solvable in polynomial time. In fact, the tractability of PW-PP(r) holds for all 2-*valued* rules, that is, positional scoring rules in which the scoring vectors contain just two distinct values. If r is the Borda rule, however, then PW-PP(r) is NP-complete. In fact, results in [1, 5, 8] imply that the possible winners problem w.r.t. the Borda rule is NP-complete, even when restricted to input partial voting profiles consisting of top-truncated ballots; therefore, the possible winners problem w.r.t. the Borda rule is NP-complete, when restricted to doubly truncated ballots. As regards partial chains, it was shown recently in [7] that the classification in Theorem 2.1 does not change for the PW-PC(r) problem. In other words, for every positional scoring rule r other than plurality and veto, PW-PC(r) is NP-complete. In particular, PW-PC(r) is NP-complete if r is the 2-approval rule or the Borda rule.

Our experimental evaluation will focus on the plurality rule, the 2-approval rule, and the Borda rule. For this reason, we summarize the aforementioned complexity results concerning these rules in the following table (also listing the veto rule for completeness).

3 NECESSARY WINNERS

Xia and Conitzer [20] presented a polynomial-time algorithm for determining whether *a particular candidate* c is a **necessary winner (NW)** in an election that uses a positional scoring rule r, that is, whether $c \in \mathrm{NW}(r, \mathbf{P})$. We recall it in Algorithm 1. We will then present several performance optimizations that allow us to efficiently compute *the set* $\mathrm{NW}(r, \mathbf{P})$ of necessary winners.

For a partial order $P \in \mathbf{P}$ and a candidate $c \in C$, we let $\textsc{Up}_P(c) = \{c' \in C | c' \succeq_P c\}$ and $\textsc{Down}_P(c) = \{c' \in C | c' \preceq_P c\}$. (Note that both $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$ include c.) Further, for a pair of candidates c and w with $c \succ_P w$, we write $\textsc{Block}_P(c, w) = \textsc{Down}_P(c) \cap \textsc{Up}_P(w)$ for the set of candidates ranked between c and w, including c and w.

Example 3.1. Consider the partial order depicted in Figure 1. If we put $c = z$, then $\textsc{Up}_P(c) = \{x, z\}$ and $\textsc{Down}_P(c) = \{z, t, u, v\}$. Next, if we put $w = v$, then $\textsc{Up}_P(w) = \{x, y, z, t, u, v\}$ and $\textsc{Down}_P(w) = \{v\}$. Thus, $\textsc{Block}_P(c, w) = \textsc{Down}_P(c) \cap \textsc{Up}_P(w) = \{z, t, u, v\}$.

Note that Algorithm 1 returns true if c is a necessary winner, not only if it's a necessary *unique* winner. To return true only if c is the necessary unique winner, line 20 should be replaced by $S(w) \geq S(c)$.

ALGORITHM 1: checkNW$(c, \mathbf{P}, r)$

for each partial order $P \in \mathbf{P}$ **do**
  for each candidate $c' \in C$ **do**
    compute $\textsc{Up}_P(c')$ and $\textsc{Down}_P(c')$
  end for
end for
for each $w \in C \setminus c$ **do**
  Initialize $S(w) = S(c) = 0$
  for each partial order $P \in \mathbf{P}$ **do**
    if $c \not\succ_P w$ **then**
      $pos_c = (m - |\textsc{Down}_P(c)| + 1)$ is the lowest possible position for c
      $pos_w = |\textsc{Up}_P(w)|$ is the highest possible position for w
      $S(c) = S(c) + r(m, pos_c)$
      $S(w) = S(w) + r(m, pos_w)$
    else if $c \succ_P w$ **then**
      slide $\textsc{Block}_P(c, w)$ between positions $|\textsc{Up}_P(w) \setminus \textsc{Down}_P(c)| + 1$ and $m - |\textsc{Down}_P(c) \setminus \textsc{Up}_P(w)|$, find positions pos_c and pos_w that minimize $r(m, pos_c) - r(m, pos_w)$
      $S(c) = S(c) + r(m, pos_c)$
      $S(w) = S(w) + r(m, pos_w)$
    end if
  end for
  if $S(w) > S(c)$ **then**
    c is not a necessary winner, return **false**
  end if
end for
c is a necessary winner, return **true**

We now present several performance optimizations that allow us to efficiently compute *the set* $\text{NW}(r, \mathbf{P})$ of necessary winners. Our optimizations are of two kinds. The first kind is based on reusing computation across candidates and on heuristically re-ordering computation. The second kind uses the structure of a given partial voting profile to optimize the computation of $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$.

Reusing and reordering computation. A straightforward way to use Algorithm 1 to compute $\text{NW}(r, \mathbf{P})$ is to execute it m times, once for each candidate.

To eliminate redundant computation, we first compute and record the $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$ of each P and c once. We will explain how to compute $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$ efficiently later in this section.

Additionally, we compute and record the best possible score $S_{max}(c) = \sum_{P \in \mathbf{P}} r(m, |\textsc{Up}_P(c)|)$ of each candidate. If a candidate $c_1 \notin \text{argmax}_c S_{max}(c)$, then that candidate is pruned as he or she cannot be a necessary winner. Indeed, for some $c_2 \in \text{argmax}_c S_{max}(c)$, there is by definition a completion in which c_2 obtains a score of $S_{max}(c_2)$. Since $S_{max}(c_2) > S_{max}(c_1)$, the score of c_1 in this completion is less than the score of c_2, and c_1 is not a winner. Hence, c_1 is not a necessary winner.

Next, we execute competitions between pairs of candidates c and w, deliberately selecting only the promising candidates as c, and prioritizing strong opponents w. Specifically, only the

candidates that have the highest $S_{max}(c)$ can become necessary winners. Further, we sort potential opponents in decreasing order of $S_{max}(w)$, because higher-scoring opponents are more likely to defeat c.

Computing $\text{UP}_P(c)$ and $\text{DOWN}_P(c)$. This part of the computation takes polynomial time, but the details of this computation are left unspecified by Xia and Conitzer. In our implementation, we use a **breadth-first search (BFS)** algorithm to compute these sets for all candidates of a given partial profile P, maintaining intermediate results in a priority queue.

We also observe that the structure of P can be used to make this computation more efficient in some common cases. In particular, $\text{UP}_P(c)$ and $\text{DOWN}_P(c)$ can be computed in $O(m)$ time for *partial chains* (recall that a partial chain is a partial order on a set C that consists of a linear order on a nonempty subset of C) and for *partitioned preferences*, where candidates are partitioned into q sets $C = A_1 \cup \cdots \cup A_q$, and where P provides a complete order over the sets but does not compare candidates within a set. A common example of partitioned preferences are top-k preferences, where the first k sets are of size 1, and the final set is of size $m-k$. Alternatively, $\text{UP}_P(c)$ and $\text{DOWN}_P(c)$ computation can be avoided altogether in these cases, since the scores of $S(c)$ and $S(w)$ that minimize $S(c) - S(w)$ can be determined directly. For example, consider a partial profile P that stores partitioned preferences, and suppose that $c \in A_i$ and $w \in A_j$. To minimize $S(c) - S(w)$, we assign c to the worst position in partition A_i, and we assign w to the best position in partition A_j.

In summary, while the optimizations described in this section do not reduce the asymptotic running time of the already polynomial $\text{NW}(r, \mathbf{P})$ computation in the general case, they are useful in practice, as we will demonstrate experimentally in Section 6.3. As we explain in the next section, we use these and similar techniques to optimize the performance of $\text{PW}(r, \mathbf{P})$, making this computation practically feasible.

4 POSSIBLE WINNERS

4.1 Computing PW for Plurality and Veto

By Theorem 2.1, for Plurality and Veto, there are polynomial-time algorithms for telling if a given candidate is a possible winner. In fact, Betzler and Dorn [3] gave such an algorithm for plurality by efficiently transforming the detection of possible winners to a network flow problem with just two layers and with integral capacities along the edges of the network. We have implemented and optimized this algorithm by, among other things, eliminating obvious winners (candidates ranked first in over half of the partial orders in $\mathbf{P}$) and obvious losers (candidates ranked first in fewer than $1/m$ partial orders), thus reducing the size of the network. A variant of this algorithm can be used to detect possible winners for veto.

4.2 Reducing PW to ILP

Again by Theorem 2.1, for all positional scoring rules other than plurality and veto, detecting possible winners is an NP-complete problem. Here, we give a polynomial-time reduction of the Possible Winners problem to ILP and, in fact, to 0-1 ILP. Let r be a positional scoring rule and let $\mathbf{s} = (s_1, \ldots, s_m)$ be its scoring vector for m candidates. Consider an input to the possible winners problem consisting of a set $C = \{c_1, \ldots, c_m\}$ of candidates, a partial voting profile $\mathbf{P} = (P_1, \ldots, P_n)$, and a distinguished candidate c_w from C; the question is whether or not $c_w \in \text{PW}(r, \mathbf{P})$.

- For each l with $1 \le l \le n$ and each i with $1 \le i \le m$, introduce m binary variables $x_1^{l,i}, x_2^{l,i}, x_3^{l,i}, \ldots, x_m^{l,i}$. Intuitively, we want to have $x_j^{l,i} = 1$ if candidate c_i has rank j in a completion T_l of P_l; otherwise, $x_j^{l,i} = 0$. Thus, the rank of c_i in T_l is equal to $\sum_{p=1}^{m} p \cdot x_p^{l,i}$.

- There are two constraints to ensure the validity of a completion T_l of P_l; namely, each candidate is assigned exactly one rank in T_l, and no two candidates are assigned the same rank in T_l:

$$\sum_{p=1}^{m} x_p^{l,i} = 1, \text{where } 1 \leq l \leq n \text{ and } 1 \leq i \leq m \tag{1}$$

$$\sum_{c_i \in C} x_p^{l,i} = 1, \text{where } 1 \leq l \leq n \text{ and } 1 \leq p \leq m. \tag{2}$$

- If a candidate c_i is ranked higher than a candidate c_j in the partial order P_l, then c_i has to also be ranked higher in a completion T_l of P_l. This is ensured by introducing the following constraint for each such pair of candidates and each partial order:

$$\sum_{p=1}^{m} p x_p^{l,j} > \sum_{p=1}^{m} p x_p^{l,i}. \tag{3}$$

- Finally, to ensure that the distinguished candidate c_w is a possible winner, we add, for each candidate $c_i \neq c_w$, the following constraint:

$$\sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot x_p^{l,i} \leq \sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot x_p^{l,w}. \tag{4}$$

Let Σ be the preceding ILP instance. Note that Σ has $O(m^2 n)$ binary variables and $O(m^2 n)$ constraints.

Note also that for the case of possible unique winners, one has $s(\mathbf{T}, c_i) < s(\mathbf{T}, c_w)$. Thus, the only change needed is to replace the inequality in Equation (4) by a strict one.

We want to show that a 0-1 solution to Σ exists if and only if candidate c_w is a possible winner. In what follows, for a set C, we let $\Pi(C)$ denote the set of all total orders on C. We also let $\pi : \Pi(C) \times C \longrightarrow [1, \ldots, m]$ be the *ranking* function that returns the rank of $c' \in C$ in a total order on C.

Theorem 4.1. *The following statements are equivalent:*

I. Candidate c_w is a possible winner w.r.t. the rule r and the partial profile $\mathbf{P}$.
II. The system Σ has a 0-1 solution.

Proof. We list four easily provable facts that will be used in the proof of the theorem. Let $C = \{c_1, \ldots, c_m\}$ and $\mathbf{P} = \{P_1, \ldots, P_n\}$ be a set of partial votes.

Fact 1. For each partial vote P_l, let $T_l \in \Pi(C)$ be a total order that extends P_l. Put

$$a_p^{l,i} = \begin{cases} 1, & \text{if } \pi(P_l, c_i) = p \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The values $a_p^{l,i}, 1 \leq i \leq m, 1 \leq p \leq m$ have the following properties:

- $\sum_{p=1}^{m} a_p^{l,i} = 1$;
- for $p = 1, \ldots, m$, we have that $\sum_{i=1}^{n} a_p^{l,i} = 1$.

Fact 2. Suppose $b_p^{l,i}$ $1 \leq i \leq n, 1 \leq p \leq m$ are non-negative integers such that

- $b_p^{l,i} \in \{0,1\}$;
- $\sum_{p=1}^{m} b_p^{l,i} = 1$;
- for $p = 1, \ldots, m$, we have $\sum_{i=1}^{n} b_p^{l,i} = 1$.

Let $\pi : \Pi(C) \times C \longrightarrow [1, \ldots, m]$ such that $\pi(T_l, c_i) = k_i$ if and only if $b_{k_i}^{l,i} = 1$. This induces a total order on C.

Fact 3. Let P_l be a partial order on C and T_l be a total order on C. Suppose that we have values $a_p^{l,i}$ as defined as in Equation (5). For all $c_i > c_j$ in P_l, the inequalities $\sum_{p=1}^{m} p(a_p^{l,j} - a_p^{l,i}) > 0$ hold if and only if $P_l \hookrightarrow T_l$.

Fact 4. Consider a profile $\mathbf{T} = (T_1, \ldots, T_n)$ and the scoring vector $\boldsymbol{s} = (s_1, \ldots, s_m)$. Note that the total score of a candidate $c_i \in C$ is $\sum_{l=1}^{n} s(T_l, c_i) = \sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot a_p^{l,i}$.

Let c_w be a fixed candidate. Then the following statements are equivalent:

- c_w is a winner in $\mathbf{T}$ using the scoring rule $\boldsymbol{s} = (s_1, \ldots, s_m)$.
- For every candidate $c_i \neq c_w$, we have that

$$\sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot a_p^{l,i} \leq \sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot a_p^{l,w}.$$

We are now ready to proceed with the proof of the theorem.

(I $\implies$ II) Assume that a partial order $P = \{P_1, \ldots, P_n\} \hookrightarrow T = \{T_1, \ldots, T_n\}$ such that c_w is a possible winner. Set

$$a_p^{l,i} = \begin{cases} 1, & \text{if } \pi(T_l, c_i) = p \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

We claim that the assignment $x_p^{l,i} \mapsto a_p^{l,i}$ satisfies all the equations of the system Σ.
Indeed, from Fact 1, we know that this assignment satisfies the following:

(1) $\sum_{p=1}^{m} a_p^{l,i} = 1$.
(2) for $p = 1, \ldots, m$, we have $\sum_{i=1}^{n} a_p^{l,i} = 1$.

Since $\mathbf{P} \hookrightarrow \mathbf{T}$, by definition, $P_l \hookrightarrow T_l$. By Fact 3, the constraints $\sum_{p=1}^{m} p(x_p^{l,j} - x_p^{l,i}) > 0$ are satisfied. Since c_w is a possible winner in P, by Fact 4, the constraints, one for each $c_i \in C \setminus \{c_w\}$,

$$\sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot x_p^{l,i} \leq \sum_{l=1}^{n} \sum_{p=1}^{m} s_p \cdot x_p^{l,w},$$

are satisfied.

(II $\implies$ I) Assume that the system Σ has the integer solution $a_p^{l,i}$ $(1 \leq l \leq n, 1 \leq i \leq m, 1 \leq p \leq m)$. Each $a_p^{l,i}$ is either 0 or 1 by the first group of constraints. Furthermore, by Fact 2, the constraints (1) and (2) ensure that each vote induces a total order $>_l$ on C. Furthermore, the total order T_l extends P_l because of the constraints $\sum_{p=1}^{m} p(a_p^{l,j} - a_p^{l,i}) > 0$. Finally, since the constraints in Equation (4) are satisfied, Fact 4 implies that c_w is a possible winner. □

4.3 Checking a Possible Winner

Determining whether or not $c_w \in \mathrm{PW}(r, \mathbf{P})$ using our methodology involves the following two main steps:

(1) Construct the ILP model. Constraints (1) and (2) depend only on m and n, whereas constraints (3) and (4) depend additionally on c_w and on the partial profile.
(2) Solve the ILP model.

Fix the values for m and n. One creates a *partial* model for the corresponding (m, n) with only constraints (1) and (2). This is called pre-processing. To save time, pre-processed models can be reused when the candidate c_w, the partial profile $\mathbf{P}$, or both change. To solve a specific problem, one loads the appropriate pre-processed model and updates it by adding constraints (3) and (4) before solving it.

4.4 Three-Phase Computation of the Set of Possible Winners

A straightforward way to compute the set of possible winners $\mathrm{PW}(r, \mathbf{P})$ is to execute the computation described in Section 4.3 above m times, once for each candidate. We now describe a more efficient method that uses pruning and early termination techniques, and heuristics to quickly identify clear possible winners. This method involves three phases, summarized here and discussed in detail below:

(1) Use $\mathrm{NW}(r, \mathbf{P})$ to identify a subset of possible winners C^1_{pw} and to prune clear non-winners C^1_{lsr}. Pass the remaining $C^1 = C \setminus (C^1_{pw} \cup C^1_{lsr})$ to the next phase.
(2) Use a heuristic to construct a completion in which $c \in C^1$ is a winner. Add all candidates for which such a completion is found to C^2_{pw}, and pass the remaining $C^2 = C^1 \setminus C^2_{pw}$ to the next phase.
(3) Invoke the subroutine described in Section 4.3 to check a possible winner for each $c \in C^2$ using an ILP solver. Add all identified possible winners to C^3_{pw}.

The final set of possible winners is $C^1_{pw} \cup C^2_{pw} \cup C^3_{pw}$.

Phase 1: Using the necessary winner algorithm. Let us denote by $S_{total}(r, m)$ the sum of scores of all candidates in some total voting profile: $S_{total}(r, m) = n \sum_{i=1}^{m} r(m, i)$. We will execute $\mathrm{NW}(r, \mathbf{P})$ to compute the set of necessary winners, which are also possible winners. Recall that as part of the $\mathrm{NW}(r, \mathbf{P})$ computation, we compute and record, for all $c \in C$, the best possible score $S_{max}(c) = \sum_{P \in \mathbf{P}} r(m, |\mathrm{UP}_P(c)|)$ of every candidate c. We can immediately identify candidates whose $S_{max}(c)$ is highest as clear possible winners, and add them to C^1_{pw}. Indeed, there is a completion $\mathbf{T}$ such that $S_{\mathbf{T}}(c) = S_{max}(c)$ and for every candidate $c' \neq c$, we have $S_{\mathbf{T}}(c') \leq S_{max}(c') \leq S_{max}(c) = S_{\mathbf{T}}(c)$. The candidate c is clearly a winner in this profile. Further, if $S_{max}(c) \geq \frac{1}{2} S_{total}(r, m)$, then c is also a possible winner and is added to C^1_{pw}.

On the other hand, if $S_{max}(c) < \frac{1}{m} S_{total}(r, m)$, then c is not a possible winner, and it can be pruned. Indeed, by the pigeonhole principle, in every completion there is a candidate with a score higher than $\frac{1}{m} S_{total}(r, m)$.

Further, consider the step in $\mathrm{NW}(r, \mathbf{P})$ where we execute competitions between pairs of candidates c and w. During a competition, we minimize the score difference $S(w) - S(c)$ over all completions. We may observe that $S(w) - S(c) > 0$. That means that the score of c will be strictly less than the score of w in every completion. This allows us to prune c as a non-winner, adding c to C^1_{lsr}.

Phase 2: Constructing a completion. Next, given a candidate c, we consider $\mathbf{P} = (P_1, \ldots, P_n)$ and heuristically attempt to create a total voting profile $\mathbf{T} = (T_1, \ldots, T_n)$ that completes $\mathbf{P}$ and in which c is the winner. If such a $\mathbf{T}$ is found, then c is added to C^2_{pw}. To construct $\mathbf{T}$, we complete each partial vote $P \in \mathbf{P}$ independently, as follows:

(1) For a given P, place c at the worst possible rank in which it achieves its best possible score. The reason for this is to minimize the scores of the items in $\text{UP}_P(c) \setminus c$.
(2) Place the remaining candidates from P into T. If multiple placements are possible, choose one that increases the score of the currently highest-scoring candidates the least.
(3) Keep a list of candidates other than c that are the possible winners so far. In subsequent completions, place these candidates as low as possible, minimizing their score.

In summary, we described a reduction of the problem of checking whether a candidate c is a possible winner to an ILP and proposed a three-phase computation that limits the number of times the ILP solver is invoked for a set of candidates C. We will show experimentally in Section 6.4 that the proposed techniques can be used to compute the set of possible winners in realistic scenarios.

5 THE REPEATED SELECTION MODEL FOR POSET GENERATION

In this section we introduce a novel generative model for partially ordered sets, called the Repeated Selection Model. It includes earlier generative models of partial orders as special cases via a suitable choice of parameters. We regard RSM as being a model of independent interest, and we also use it here as part of our experimental evaluation, described in Section 6. To start, we introduce the **Repeated Insertion Model (RIM)** that is used for generating total orders in Section 5.1. We then describe our novel RSM model in Section 5.2.

5.1 Preliminaries: The Repeated Insertion Model (RIM)

In this section we represent total orders using rankings, that is, ordered lists of items indexed by position. We will use $\boldsymbol{\sigma}$, $\boldsymbol{\tau}$, and so on to denote rankings. We will use $\boldsymbol{\sigma}(i)$ to refer to an item at position i in $\boldsymbol{\sigma}$, and we will use $\boldsymbol{\sigma}^{-1}(a)$ to denote the position of element a in $\boldsymbol{\sigma}$. When describing iterative algorithms, for convenience of presentation we will denote by $\boldsymbol{\sigma}_i$ the value of $\boldsymbol{\sigma}$ at step i.

The RIM is a generative model that defines a probability distribution over rankings due to Doignon et al. [9]. This distribution, denoted by $\text{RIM}(\boldsymbol{\sigma}, \Pi)$, is parameterized by a reference ranking $\boldsymbol{\sigma}$ and a function Π, where $\Pi(i, j)$ is the probability of inserting $\boldsymbol{\sigma}(i)$ at position j. Here, Π is a matrix where each row corresponds to a valid probability distribution (i.e., the values in a row sum up to one). Algorithm 2 presents the RIM sampling procedure. It starts with an empty ranking $\boldsymbol{\tau}$, inserts items in the order of $\boldsymbol{\sigma}$, and puts item $\boldsymbol{\sigma}(i)$ at the j^{th} position of the currently incomplete $\boldsymbol{\tau}$ with probability $\Pi(i, j)$. The algorithm terminates after m iterations and outputs $\boldsymbol{\tau}$, a total order over the items drawn from $\boldsymbol{\sigma}$.

ALGORITHM 2: $\text{RIM}(\boldsymbol{\sigma}, \Pi)$

```
Initialize an empty ranking τ = ⟨⟩.
for i = 1, ..., m do
    Select a random position j ∈ [1, i] with a probability Π(i, j)
    Insert σ(i) into τ at position j
end for
return τ
```

Example 5.1. $\text{RIM}(\langle a, b, c\rangle, \Pi)$ generates $\boldsymbol{\tau} = \langle b, c, a\rangle$ as follows:

- Initialize an empty ranking $\boldsymbol{\tau}_0 = \langle\rangle$.
- At step 1, $\boldsymbol{\tau}_1 = \langle a\rangle$ by inserting a into $\boldsymbol{\tau}_0$ with probability $\Pi(1, 1) = 1$.
- At step 2, $\boldsymbol{\tau}_2 = \langle b, a\rangle$ by inserting b into $\boldsymbol{\tau}_1$ at position 1 with probability $\Pi(2, 1)$.
- At step 3, $\boldsymbol{\tau} = \langle b, c, a\rangle$ by inserting c into $\boldsymbol{\tau}_2$ at position 2 with probability $\Pi(3, 2)$.

The overall probability of sampling τ is $\Pr(\tau \mid \langle a, b, c\rangle, \Pi) = \Pi(1,1) \cdot \Pi(2,1) \cdot \Pi(3,2)$. Note that this particular sequence of steps is the only way to sample $\langle b, c, a\rangle$ from $\mathsf{RIM}(\langle a, b, c\rangle, \Pi)$.

The Mallows model [16], $\mathsf{MAL}(\sigma, \phi), \phi \in (0, 1]$, is a special case of RIM. As a popular preference model, it defines a distribution of rankings that is analogous to the Gaussian distribution: the ranking σ is at the center, and rankings closer to σ have higher probabilities. Specifically, the probability of a ranking τ is given by

$$\Pr(\tau|\mathsf{MAL}(\sigma,\phi)) = \frac{\phi^{dist(\sigma,\tau)}}{1 \cdot (1+\phi) \cdot (1+\phi+\phi^2) \ldots (1+\cdots+\phi^{m-1})}. \tag{7}$$

Here, $dist(\sigma, \tau)$ is the Kendall-tau distance between σ and τ: $dist(\sigma, \tau) = |(a, a') \mid a \succ_\sigma a', a' \succ_\tau a|$, that is, the number of preference pairs (a, a') that appear in the opposite relative order in σ and τ. The expression in the denominator of Equation (7) is the normalization constant, which we will find convenient to denote $Z_{\phi,m}$. When $\phi \to 0$, the probability mass is concentrated around the reference ranking σ; when $\phi = 1$, all rankings have the same probability; that is, $\mathsf{MAL}(\sigma, 1)$ is the uniform distribution over rankings.

As was shown in [9], $\mathsf{RIM}(\sigma, \Pi)$ is precisely $\mathsf{MAL}(\sigma, \phi)$ when $\Pi(i, j) = \frac{\phi^{i-j}}{1+\phi+\cdots+\phi^{i-1}}$. That is, the Mallows model is a special case of RIM, and so RIM can be used as an efficient sampler for Mallows.

5.2 The Repeated Selection Model (RSM)

The RSM is a generative model that defines a probability distribution over posets. Intuitively, in this model we iteratively select a random item and randomly choose whether it succeeds each of the remaining items. More formally, an instance of this distribution, denoted, $\mathsf{RSM}(\sigma, \Pi, p)$, is parameterized by three elements:

- a reference ranking σ of length m,
- a selection probability function Π, where $\Pi(i, j)$ is the probability of *selecting* the j^{th} item among the remaining items at step i, and
- a preference probability function $p : \{1, \ldots, m-1\} \to [0, 1]$ that determines the probability $p(i)$ that the i^{th} selected item precedes (is preferred to) each of the remaining items.

We view Π as a matrix where each row corresponds to a valid probability distribution (i.e., the values in a row sum up to one) and the $i-1$ rightmost entries in the ith row are zero (i.e., for all i and for all $j > m - i + 1$, we have $\Pi(i, j) = 0$).

Example 5.2. An example is the following combination of parameters:

$$\sigma = \langle a, b, c\rangle$$

$$\Pi = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.6 & 0.4 & 0 \\ 1.0 & 0 & 0 \end{pmatrix}$$

$$p(1) = 0.3 \quad p(2) = 0.4$$

Algorithm 3 presents the RSM sampling procedure. At each step, we select a random candidate c and remove it from σ. Hence, when we begin the ith step, there are $m - i + 1$ remaining candidates in σ. The choice of c in the ith step is via the probability distribution in the i^{th} row of the selection probability matrix Π (line 4 of Algorithm 3). Once c is selected, RSM decides whether to add a preference $c \succ d$, where d is a candidate from the remaining σ, with probability $p(i)$; note that decisions of $c \succ d$ are probabilistically independent for different candidates d (lines 6–8 of Algorithm 3). Note that the preferences are always made from candidates selected earlier to candidates selected later; hence, the resulting set of preferences induces a valid partial order.

ALGORITHM 3: RSM($\boldsymbol{\sigma}, p, \Pi$)

Initialize an empty poset $\boldsymbol{\tau} = \langle\rangle$.
for $i = 1, \ldots, m-1$ **do**
  Select a random position $j \in [1, m-i+1]$ with a probability $\Pi(i, j)$
  Select candidate $c = \boldsymbol{\sigma}(j)$
  Remove c from $\boldsymbol{\sigma}$ (which now contains $|\boldsymbol{\sigma}| = m - i$ candidates)
  for $k = 1, \ldots, m-i$ **do**
    Add the pair $c \succ \boldsymbol{\sigma}(k)$ to $\boldsymbol{\tau}$ with probability $p(i)$ (or leave it out with probability $1 - p(i)$)
  end for
end for
return the transitive closure of $\boldsymbol{\tau}$

Example 5.3. Continuing Example 5.2, the model RSM($\langle a, b, c\rangle, \Pi, p$) can generate the poset $\boldsymbol{\tau} = \langle b \succ a, a \succ c, b \succ c\rangle$ as follows:

- Initialize an empty poset $\boldsymbol{\tau} = \langle\rangle$.
- At step $i = 1$, select b with probability $\Pi(1, 2) = 0.3$ and remove it from $\boldsymbol{\sigma}$, setting $\boldsymbol{\sigma}_1 = \langle a, c\rangle$. Then, add the pair $b \succ a$ to $\boldsymbol{\tau}$ with probability $p(1) = 0.3$, and *do not add* the pair $b \succ c$ to $\boldsymbol{\tau}$ with probability $1 - p(1) = 0.7$.
- At step $i = 2$, select a with probability $\Pi(2, 1) = 0.6$ and remove it from $\boldsymbol{\sigma}$, setting $\boldsymbol{\sigma}_2 = \langle c\rangle$. Finally, add the pair $a \succ c$ to $\boldsymbol{\tau}$ with probability $p(2) = 0.4$.
- Take the transitive closure of $\boldsymbol{\tau}$ and return $\langle b \succ a, a \succ c, b \succ c\rangle$.

The probability of sampling $\boldsymbol{\tau}$ in this way is $\Pi(1, 2) \cdot p(1) \cdot (1 - p(1)) \cdot \Pi(2, 1) \cdot p(2) = 0.01512$.

Note that the same $\boldsymbol{\tau}$ can be generated by RSM($\langle a, b, c\rangle, \Pi, p$) using a different sequence of steps, thus yielding a different probability. In our example there is one other way to derive $\boldsymbol{\tau}$: at step $i = 1$, add $b \succ c$ to $\boldsymbol{\tau}$ with probability $p(1)$. This happens with the probability $\Pi(1, 2) \cdot p(1) \cdot p(1) \cdot \Pi(2, 1) \cdot p(2)$. These are the only two possible derivations of $\boldsymbol{\tau}$ in our example. These yield the total probability

$$\Pr(b \succ a \succ c \mid \langle a, b, c\rangle, \Pi, p) = \Pi(1, 2) \cdot p(1) \cdot \Pi(2, 1) \cdot p(2) = 0.0216.$$

In the general case, however, it is not clear whether this probability can be computed efficiently. In particular, the probability of a poset may be due to all the linear extensions of the poset.

Importantly, RSM includes several generative models of partial orders as special cases via a suitable choice of parameters. For example, $p = (1, \ldots, 1, 0, \ldots, 0)$ with k ones will generate a top-truncated partial order, whereas $p = (0, \ldots, 0, 1, \ldots, 1)$ with k ones will generate a partial chain over a subset of k items. Moreover, a uniform p gives rise to the generative model referred to as *Method 1* of Gehrlein [10]. Figure 2 compares these probability distributions empirically. Finally, as we show below, the Mallows model is also a special case of RSM.

Theorem 5.4. *For a given $\phi \in (0, 1]$, and for $p(i) = 1$ for all i, we have that* RSM($\boldsymbol{\sigma}, \Pi, p$) *is precisely* MAL($\boldsymbol{\sigma}, \phi$)*, when for all i and for all j, we have that $\Pi(i, j) = \frac{\phi^{j-1}}{\sum_{k=1}^{m-i+1} \phi^{k-1}}$, if $j \le m - i + 1$, and $\Pi(i, j) = 0$, if $j > m - i + 1$.*

Proof. First, observe that because $p(i) = 1$ for all i, RSM($\boldsymbol{\sigma}, \Pi, p$) will generate total orders. Further, observe that, although an arbitrary poset can be generated by RSM in multiple ways (as demonstrated by Example 5.3), there is only one way to obtain a total order (ranking). We will show by induction on the number of candidates m in $\boldsymbol{\sigma}$ that $\Pr(\boldsymbol{\tau} \mid \text{RSM}(\boldsymbol{\sigma}, \Pi, p)) = \Pr(\boldsymbol{\tau} \mid$

$MAL(\boldsymbol{\sigma}, \phi)$). Recall from Equation (7) that $\Pr(\boldsymbol{\tau} \mid MAL(\boldsymbol{\sigma}, \phi)) = \phi^{dist(\boldsymbol{\sigma}, \boldsymbol{\tau})}/Z_{\phi,m}$, where $Z_{\phi,m}$ is a normalization constant, and $dist(\boldsymbol{\sigma}, \boldsymbol{\tau})$ is the Kendall-tau distance between $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$: $dist(\boldsymbol{\sigma}, \boldsymbol{\tau}) = |(a, a') \mid a \succ_{\boldsymbol{\sigma}} a', a' \succ_{\boldsymbol{\tau}} a|$, that is, the number of preference pairs (a, a') that appear in the opposite relative order in $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$. For notational convenience, we will denote by $\boldsymbol{\sigma}_{-a}$ a subranking of $\boldsymbol{\sigma}$ with item a removed. Further, we will denote by $\Pi_{-i,-j}$ a projection of the matrix Π with the i^{th} column and j^{th} row removed.

Base case. When $m = 1$, both RSM and Mallows generate a single ranking with probability 1.

Inductive step. Suppose that RSM and Mallows assign the same probability to the subranking of some $\boldsymbol{\tau}$ with the first element $\boldsymbol{\tau}(1)$, denoted τ_1, removed, and with $\boldsymbol{\sigma}$ and Π adjusted accordingly:

$$\Pr(\boldsymbol{\tau}_{-\tau_1} \mid \mathsf{RSM}(\boldsymbol{\sigma}_{-\tau_1}, \Pi_{-1,-m}, p)) = \Pr(\boldsymbol{\tau}_{-\tau_1} \mid \mathsf{MAL}(\boldsymbol{\sigma}_{-\tau_1}, \phi)) = \frac{\phi^{dist(\boldsymbol{\sigma}_{-\tau_1}, \boldsymbol{\tau}_{-\tau_1})}}{Z_{\phi,m-1}}. \tag{8}$$

Let us now consider the ranking $\boldsymbol{\tau}$ of length m, and observe that $dist(\boldsymbol{\tau}, \boldsymbol{\sigma}) = dist(\boldsymbol{\tau}_{-\tau_1}, \boldsymbol{\sigma}_{-\tau_1}) + \boldsymbol{\sigma}^{-1}(\tau_1) - 1$, where $\boldsymbol{\sigma}^{-1}(\tau_1)$ is the position of element τ_1 in $\boldsymbol{\sigma}$. Moreover, the probability to select τ_1 at the first step of RSM is given by

$$\Pi(1, \sigma^{-1}(\tau_1)) = \frac{\phi^{\sigma^{-1}(\tau_1)-1}}{\sum_{k=1}^{m} \phi^{k-1}}. \tag{9}$$

Combining Equations (8) and (9), and recalling the expression for $Z_{\phi,m}$ from Equation (7), we obtain the following probability for $\boldsymbol{\tau}$:

$$\begin{aligned}\Pr(\boldsymbol{\tau} \mid \mathsf{RSM}(\boldsymbol{\sigma}, \Pi, p)) &= \Pi(1, \sigma^{-1}(\tau_1)) \times \Pr(\boldsymbol{\tau}_{-\tau_1} \mid \mathsf{RSM}(\boldsymbol{\sigma}_{-\tau_1}, \Pi_{-1,-m}, p)) \\ &= \frac{\phi^{\sigma^{-1}(\tau_1)-1}}{\sum_{k=1}^{m} \phi^{k-1}} \times \frac{\phi^{dist(\boldsymbol{\sigma}_{-\tau_1}, \boldsymbol{\tau}_{-\tau_1})}}{Z_{\phi,m-1}} = \frac{\phi^{dist(\boldsymbol{\tau}, \boldsymbol{\sigma})}}{Z_{\phi,m}} = \Pr(\boldsymbol{\tau} \mid \mathsf{MAL}(\boldsymbol{\sigma}, \phi)).\end{aligned}$$

The proof by induction concludes and the theorem is proven. □

6 EXPERIMENTAL EVALUATION

All experiments were carried out on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, with 412 GB of RAM, 20 hyper-threaded cores running two threads per core, running Ubuntu 16.04.6 LTS. We used Python 3.5 for our implementation, and the solver Gurobi v8.1.1 [11] for solving the ILP instances produced by the reduction from instances of the possible winner problem.

6.1 Experimental Datasets and Scoring Rules

Real datasets. We used two real datasets in our experimental evaluation, *travel* and *dessert*.

The Google Travel Review Ratings dataset (*travel*) [18] consists of average ratings (each between 1 and 5) issued by 5,456 users for up to 24 travel categories in Europe. For each user, we create a set of preference pairs such that items in each pair have different ratings (no tied pairs). Items for which a user does not provide a rating are not included into that user's preferences. Because preferences are derived from ratings issued by individual users, there cannot be any cycles in the set of preference pairs corresponding to a given user.

The *dessert* dataset was collected by us. It consists of user preferences over pairs of eight desserts, collected from 228 users, with up to 28 pairwise judgments per user. For each pair, users indicated their confidence in the preference using a sliding bar. This enabled us to create several voting profiles based on this data, each corresponding to a particular confidence threshold. With a high confidence threshold, we keep fewer pairs and obtain a sparse profile, and with a low confidence threshold, we keep more pairs and obtain a very dense profile. Because preferences are collected

pairwise, there can be cycles. We check the set of preferences of each user and only keep those that are acyclic for the experiments in this article, discarding all preferences of users whose preferences contain cycles.

Synthetic datasets. We use three different types of synthetic voting profiles, namely, *partial chains*, *partitioned preferences*, and *RSM Mix* . We now describe the data generation process for each.

Recall from Definition 2.2 in Section 2 that a *partial chain* on a set C is a partial order on C that consists of a linear order on a non-empty subset C' of C. Further, recall that a *partitioned preference* on a set C is a partial order on C with the property that C is partitioned into disjoint subsets $A_1, \ldots, A_q$ such that (a) every element from A_i is preferred to every element from A_j, for $i < j \leq q$, and (b) the elements in each A_i are pairwise incomparable.

We are given the set of candidates $C = \{c_1, c_2, c_3, \ldots, c_m\}$ and the number of voters n. To generate a partial chains profile or a partitioned preferences profile, we start with a mixture of three Mallows models, each with $\phi = 0.5$, with a randomly chosen σ of size m, and covering approximately $\frac{1}{3}$ of the voters, and generate a complete voting profile $\mathbf{T} = (T_1, \ldots, T_n)$ of total orders on C. Then, to generate a partial chains profile, for each total order T_i, choose $d \in [0, m-2]$ uniformly at random, and drop d candidates from T_i by selecting one uniformly at random from the remaining candidates over d iterations. To generate a partitioned preferences profile, for each T_i, choose the number q of non-empty partitions uniformly at random from the set $[2, m]$. To partition T_i, select $q-1$ positions between 2 and m uniformly at random without replacement, with each position corresponding to the start of a new partition. Drop the order relations between candidates in the same partition.

To generate an *RSM Mix* voting profile, we use a mixture of three RSMs (as described in Section 5), each covering $\frac{1}{3}$ of the voters, with selection probability $\Pi_{i,j}$ corresponding to the Mallows model ($\phi = 0.5$, randomly chosen σ of size m). For each of the three RSMs, we draw the preference probability $p(m)$ uniformly from $[0, 1]$ for each m.

We use a mixture of three Mallows models, each covering approximately $\frac{1}{3}$ of the voters as there is no significant difference (Student's t-test, $p > 0.05$) between the running times with a change in the number of Mallows models. To validate this, we generated five partial chain profiles, each using a mixture of three, six, and nine Mallows models, with $m = 25$ candidates and $n = 10{,}000$ voters, and used the Borda scoring rule. We did the three-phase computation of possible winners on these datasets. Similarly, there is no significant difference (Student's t-test, $p > 0.05$) between running times when RSM Mix is generated with a mixture of three, six, and nine RSMs using the same setup. Overall, there is no effect of varying the number of Mallows models or RSMs on the running times.

There is also no significant difference (Student's t-test, $p > 0.05$) between the running times with a change in the distribution of the proportion of voters each model covers when using a mixture of three Mallows models. To validate this, we generated five partial chain profiles, each using a mixture of three Mallows models to create equally divided, evenly peaked, and unevenly peaked datasets, with $m = 25$ candidates and $n = 10,000$ voters, and used the Borda scoring rule. Equally divided, evenly peaked, and unevenly peaked datasets distribute the voters as follows: {33%, 33%, 33%}, {25%, 50%, 25%}, and {30%, 60%, 10%}, respectively. We did the three-phase computation of possible winners on these datasets. Similarly, there is no significant difference (Student's t-test, $p > 0.05$) between running times when RSM Mix is generated using a mixture of three RSMs, with each RSM covering a varying proportion of voters as discussed.

Scoring rules. We evaluated the performance of our techniques for three positional scoring rules, namely, the plurality rule, the 2-approval rule, and the Borda rule. We chose these rules for two

Table 1. Complexity of the Possible Winners (PW) and Necessary Winners (NW) Problems, and Their Restrictions to Partitioned Preferences (PW-PP), Top-Truncated Ballots (PW-TTB), and Partial Chains (PW-PC) w.r.t. plurality, veto, 2-Approval, and Borda Rules

Scoring Rule	PW	PW-PP	PW-TTB	PW-PC	NW (All Kinds)
Plurality & Veto	P	P	P	P	P
2-Approval	NP-complete	P	P	NP-complete	P
Borda	NP-complete	NP-complete	NP-complete	NP-complete	P

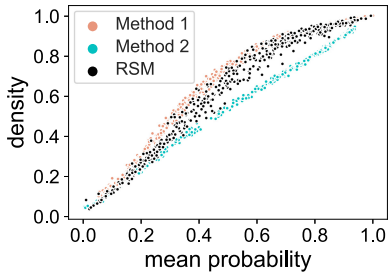


Fig. 2. Density distribution of RSM, compared to two poset generation methods from Gehrlein [10].

reasons. First, they are arguably among the most well-known and extensively studied positional scoring rules. Second, the plurality rule and the 2-approval rule are prototypical examples of *bounded-value* rules, that is, rules in which the scores are of bounded size (in this case, the bound on the size is 2), while the Borda rule is a prototypical example of an *unbounded-value* rule; that is, the scores may grow beyond any fixed bound. Note that we also conducted experiments for the veto rule and found out that performance followed the same trends as those for plurality. We remind the reader that the results about the complexity of the necessary winners and the possible winners with respect to the plurality rule, the 2-approval rule, and the Borda rule are summarized in Table 1 in Section 2.

6.2 Validation of the Repeated Selection Model (RSM)

In this section we compare the RSM with *Method 1* and *Method 2* from Gehrlein [10]. Our first comparison is of the empirical distribution of poset density, defined as $d = \frac{D}{\binom{m}{2}} = \frac{2D}{m(m-1)}$, where m is the number of items, and D is the total number of preference pairs in the partial voting profile $\mathbf{P}$. Figure 2 presents this comparison for 10 candidates and 50 voters. We observe that the RSM generates partial orders over a wider range of densities than either of the two methods from Gehrlein.

We also conducted an experiment to verify that RSM is sufficiently flexible to represent real partial voting profiles. To do this, we extended the methods of Stoyanovich et al. [19] to fit a single RSM to the *dessert* and *travel* datasets and compared the goodness of fit to that of Method 1 and Method 2 from Gehrlein [10]. For each dataset, we compute the negative log-likelihood using the voters for whom we know both the real subranking $\boldsymbol{\tau}$ and the synthetically generated subranking $\boldsymbol{\tau}'$:

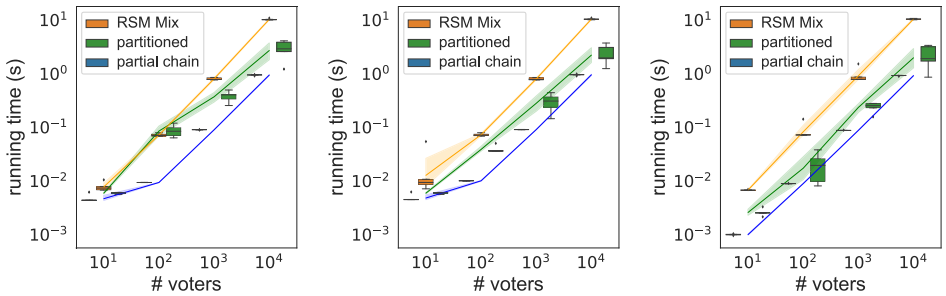
$$\mathcal{NLL} = -\frac{1}{n}\sum_{i}^{n} log(\Pr(\boldsymbol{\tau}'_i \mid \boldsymbol{\tau}_i)).$$

Our results are summarized in Table 2 and confirm that RSM fits these real datasets more closely than other methods, as quantified by negative log-likelihood ($\mathcal{NLL}$), with lower values

Table 2. Comparison of Goodness-of-Fit of RSM (Section 5), and Methods 1 and 2 from Gehrlein [10], on Real-world Datasets

Dataset	$\mathcal{NLL}$		
	RSM	*Method 1*	*Method 2*
Travel	**9.5**	10.4	10.4
Dessert (Sparse)	**12.5**	12.9	14.7
Dessert (Dense)	**18.7**	19.1	19.8

$\mathcal{NLL}$ stands for negative log-likelihood, with lower values corresponding to better fit.



(a) plurality (b) 2-approval (c) Borda

Fig. 3. Running time of the **necessary winners** computation for 100 candidates and between 10 and 10,000 voters, for three positional scoring rules.

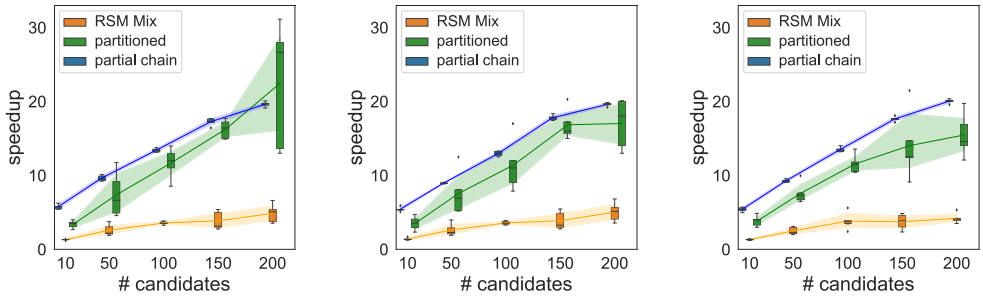
corresponding to better fit. Note that all methods fit the *travel* dataset better as compared to the *dessert* dataset because the former contains partitioned preferences with missing candidates.

6.3 Necessary Winners

In this section, we evaluate the performance of an optimized version of the polynomial-time algorithm by Xia and Conitzer [20], as described in Section 3 for three positional scoring rules: plurality, 2-approval, and Borda.

We start with experiments that demonstrate the impact of the number of voters n and the number of candidates m on the running time of the optimized necessary winners algorithm described in Section 3. In Figure 3, we set $m = 100$, vary n between 10 and 10,000 on a logarithmic scale, and show the running time for each of the rules, plurality, 2-approval, and Borda, and for each family of synthetic datasets as a box-and-whiskers plot. We observe that the computation is efficient: RSM Mix is the most challenging and completes in 10 seconds or less for 10,000 voters across all scoring rules. The running time increases linearly with n.

Next, we analyzed the speed-up achieved by the optimized necessary winners algorithm for $n = 10{,}000$ voters, with m ranging from 10 to 200 on a linear scale. Figure 4 shows these results in comparison to a baseline, where we reuse computation of $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$ across candidates but do not re-order candidates in a competition, and also do not optimize the computation of $\textsc{Up}_P(c)$ and $\textsc{Down}_P(c)$ based on the structure of $\mathbf{P}$. We observe that the optimized implementation outperforms the baseline by a factor of 10 to 20 in most cases. Overall, speed-up improves with increasing number of candidates, and partial chains and partitioned preferences datasets show the



(a) plurality (b) 2-approval (c) Borda

Fig. 4. Speed-up factor of the optimized **necessary winners** computation over the baseline, for 10,000 voters, and between 10 and 200 candidates, for three positional scoring rules. Performance is improved by a factor of 10 to 20 in most cases and is highest for partial chains and partitioned preferences.

highest speed-up. We see significant variability in Figure 4(a), because some of the instances had necessary winners and others did not.

We also analyzed the running time and observed that this computation is efficient: RSM Mix completes in under 40 seconds for $m = 200$ for plurality and 2-approval, and for all except one case of Borda, where it takes 60 seconds. For partitioned preferences and partial chains, the computation completes in under 8.5 seconds and 2 seconds, respectively, pointing to the effectiveness of the optimizations that use the structure of P.

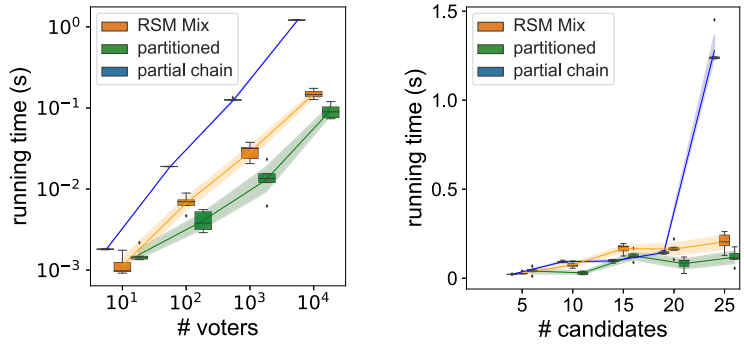
Finally, the running times were interactive on real datasets: 0.006 seconds for all scoring rules on *dessert* and 0.28 seconds on *travel*. We achieved a factor of 2 to 2.5 speed-up over the baseline version for *dessert*, and a factor of 5 to 6.7 speed-up for *travel*. Speed-up was most significant for Borda, with running time decreasing from 1.87 seconds to 0.28 seconds.

6.4 Possible Winners

In this section, we evaluate the performance of appropriate methods for the computation of **possible winners (PW)** under plurality, 2-approval, and Borda.

Plurality. To compute PW under plurality, we implemented an optimized version of the polynomial-time algorithm by Betzler and Dorn [3], as described in Section 4.1. Figure 5 shows the running time of our implementation. In Figure 5(a), we set $m = 25$ and vary n between 10 and 10,000 on a logarithmic scale, while in Figure 5(b), we set $n = 10,000$ and vary m between 5 and 25 on a linear scale. We observe that this algorithm is efficient: most instances complete in less than 0.5 second, with the exception of a single instance that takes just over 1 second. The running time is higher when there are more possible winners. For this reason, computation is fastest on partitioned preferences datasets and slowest on partial chains datasets. (Note that we set m lower for PW experiments than for NW, where m went up to 200, to have the same experimental setting for plurality as for 2-approval and Borda, presented later in this section. A high value of m is infeasible for the latter rules because of the intrinsic complexity of the problem.)

2-Approval and Borda using three-phase computation. As discussed in Section 2, computing PW under the Borda rule is NP-complete both for voting profiles consisting of partial chains and for voting profiles consisting of partitioned preferences. Furthermore, computing PW under 2-approval is NP-complete for voting profiles consisting of partial chains but is polynomial-time solvable for voting profiles consisting of partitioned preferences. In view of the intractability implied by the



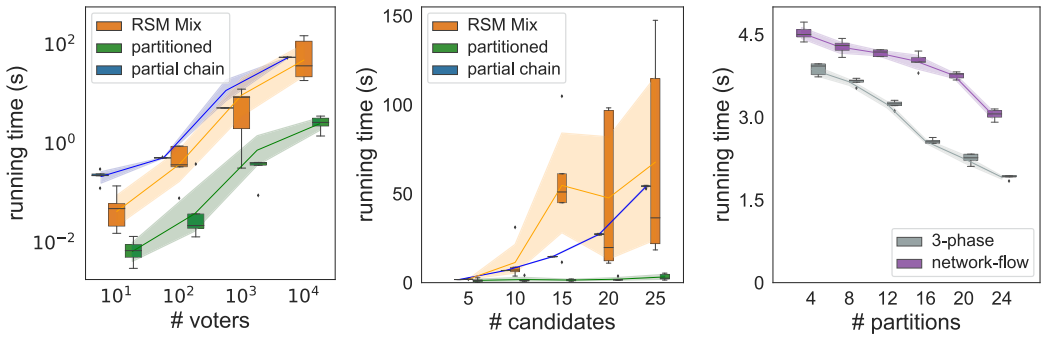
(a) 25 candidates (b) 10,000 voters

Fig. 5. Running time of the computation of the set of **possible winners** on **plurality**, using an optimized implementation of the algorithm from Betzler and Dorn [3]. Most instances complete in less than 0.5 seconds. Running times are higher when there are more possible winners. For this reason, computation is fastest on partitioned preferences and slowest on partial chains.

aforementioned NP-complete cases, we use the three-phase method described in Section 4.4 that may invoke the ILP solver for difficult cases. We evaluate the performance of this method here, demonstrating the impact of the number of voters n and the number of candidates m on the running time of PW. (Note that we include 2-approval for partitioned preferences into the comparison, for consistency of presentation.) We fix m at 25 and vary n between 10 and 10,000 on a logarithmic scale, and then fix n at 10,000 and vary m between 5 and 25 on the linear scale. Even with these modest values of m, the ILP solver can take a very long time. Thus, to make our experimental evaluation manageable, we set an end-to-end cut-off of 2,000 seconds per instance. In what follows, we report the running times of the instances that completed within the cut-off and additionally report the percentage of completed instances.

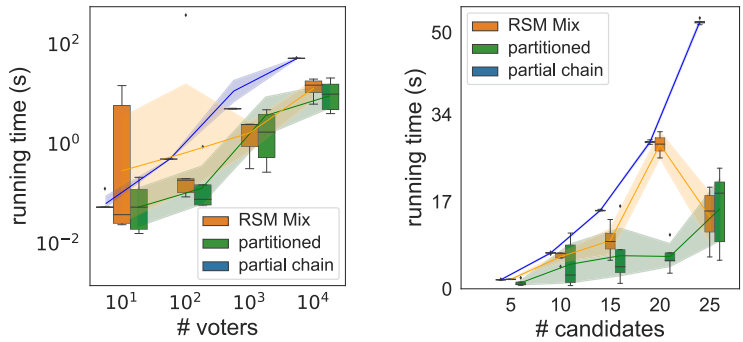
Figures 6(a) and 7(a) show the running time as a function of the number of voters for 2-approval and Borda, respectively. The running time increases linearly with the number of voters. Interestingly, partial chains datasets take as long or longer to process as RSM Mix datasets. This is because Phase 1 of the three-phase computation is more effective for RSM Mix, with fewer candidates passed on to Phase 2. Phases 1 and 2 are effective in pruning non-winners and in identifying clear possible winners. Of the 75 instances we executed for this experiment for each scoring rule, only 13 (17%) needed to execute Phase 3 (i.e., invoke the ILP solver) for 2-approval, and only 9 (12%) for Borda, with at most three candidates to check. Of the nine instances that reached Phase 3 for Borda, six timed out at 2,000 seconds. No other instances timed out in this experiment. Instances reaching Phase 3 are responsible for the high variability in the running times. For 2-approval, all instances that reached Phase 3 computed in under 148 seconds (median 8.46 seconds, mean 30.69 seconds, stdev 48.72 seconds). For Borda, for the three instances that executed Phase 3 and did not time out, the running times were 6 seconds, 15 seconds, and 381 seconds. In contrast, all remaining instances—those that did not execute Phase 3—computed in under 53 seconds (median 2.44 seconds, mean 9.50 seconds, stdev 15.14 seconds).

Figures 6(b) and 7(b) show the running times as a function of the number of candidates under 2-approval and Borda. We make similar observations here as in our discussion of Figures 6(a) and 7(a), noting that only nine instances reached Phase 3 for 2-approval, and only four reached Phase 3 for Borda. These instances, all RSM Mix, took longer to run and contributed the most to running time variability.



(a) 25 candidates (b) 10,000 voters (c) 25 candidates & 10,000 voters

Fig. 6. Running time of the three-phase computation of the set of **possible winners** on **2-approval** scoring rule. (a) None of the 75 instances timed out at 2,000 seconds. These were among only 13 instances (9 RSM Mix and 4 partial chain) that needed to execute the ILP solver in Phase 3. (b) None of the 75 instances timed out at 2,000 seconds. These were among only nine instances (all RSM Mix) that needed to execute the ILP solver in Phase 3. (c) Comparison of three-phase computation with polynomial-time algorithm (based on flow network and theoretical results in [12]) by varying the number of partitions in each instance.



(a) 25 candidates (b) 10,000 voters

Fig. 7. Running time of the three-phase computation of the set of **possible winners** on **Borda** scoring rule. (a) Six (all RSM Mix) out of 75 instances (8.0%) timed out at 2,000 seconds. These were among only nine instances (all RSM Mix) that needed to execute the ILP solver in Phase 3. (b) Four (all RSM Mix) out of 75 instances (5.3%) that needed to execute the ILP solver timed out at 2,000 seconds.

2-Approval on partitioned preferences: three-phase computation vs. network flow. For the 2-approval rule on voting profiles consisting of partitioned preferences, we also implemented Kenig's [12] polynomial-time algorithm, which is based on network flow, and we compared its performance to that of three-phase computation. Figure 6(c) shows the running times as a function of the number of partitions for instances containing 25 candidates and 10,000 voters. None of the 30 instances needed ILP (Phase 3) while using the three-phase computation. Overall, our three-phase approach is more general in terms of the datasets it handles, and it outperforms the polynomial-time network-flow algorithm for partitioned preferences.

Drilling down on the phases of the three-phase computation. Now, we drill down on each phase of the three-phase computation to study the effectiveness of each phase:

Table 3. Percentage of Candidates Pruned at the End of Each Phase of the Three-phase Computation (Cumulative Percentages Are Written in the Brackets)

Dataset	Rule	Phase 1	Phase 2	Phase 3 (ILP)
Partial chain	*2-Approval*	22.9%	77% (99.9%)	0.1% (100%)
	Borda	18.3%	81.7% (100%)	
Partitioned	*2-Approval*	98.3%	1.7% (100%)	
	Borda	64.8%	35.1% (99.9%)	0.1% (100%)
RSM Mix	*2-Approval*	63.4%	33.1% (96.5%)	3.5% (100%)
	Borda	59.4%	39.4% (98.8%)	1.2% (100%)

1. Effectiveness of each phase: We measured the effectiveness of each phase of the three-phase computation (Table 3) by measuring the percentage of candidates pruned at the end of each phase. We observed that on average, 99.18% of the candidates are pruned after the first two phases that run in time polynomial in the number of candidates. Specifically, we make two key observations: (i) Phase 1 (reusing the NW algorithm) is most effective when there are few possible winners, and (ii) RSM Mix is the toughest dataset to handle as it needs Phase 3 (ILP) the most.

2. Effectiveness of the polynomial-time computation of the first two phases: Next, we specifically measured the effectiveness of the first two phases of the three-phase computation, which run in polynomial time in the number of candidates, on a larger number of RSM profiles. To do so, we calculate the proportion of profiles for which the three-phase computation terminates after the first two phases, under the Borda scoring rule. We created 10,000 profiles consisting of 100 voters and 10 candidates using a mixture of three RSMs, as described in Section 6.1.

Figure 8 presents the density distribution of the resulting posets (as in Figure 2) and highlights the instances for which PW computation terminated after two phases in purple and those for which all three phases were necessary in yellow. In summary, PW terminated after two phases for 91.62% of the instances. Phase 3 was needed primarily when the ϕ parameter was low and the average density was medium, or when the ϕ parameter and the average density were both high. Profiles with low average density always terminated after the second phase.

3. Effectiveness across different data generation methods: We also compared the average running time of the first two phases of the PW algorithm using RSM profiles with profiles generated using Gehrlein's methods. In this experiment, we generated 10,000 profiles using a mixture of three RSM models, with $m = 10$ candidates and $n = 100$ voters, and used the Borda scoring rule. RSM profiles generally take more time across different values of ϕ (Figure 9(a)) and across different poset densities (Figure 9(b)) as RSM is more generalized (Figure 2). This finding once again underscores that RSM is able to generate interesting posets, which may be more challenging to process than those generated with alternative methods. In addition, it also underscores the overall effectiveness of our three-phase computation, irrespective of the method used.

PW on real datasets. Finally, we computed PW for the real datasets *dessert* and *travel* using three-phase computation and found multiple possible winners for all scoring rules. In all cases, winners were determined in Phases 1 and 2 of the computation, and the ILP is never invoked. All executions took under 23 seconds.

7 CONCLUDING REMARKS

The contributions made in this article can be summarized as follows:

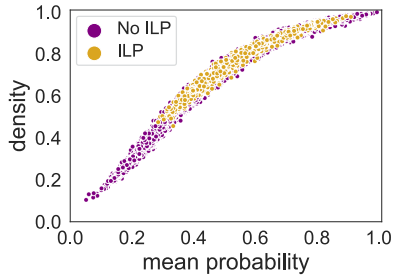


Fig. 8. 91.62% of 10,000 RSM profiles, with 10 candidates and 100 voters, found the entire set of possible winners under the Borda scoring rule after the first two phases of pruning without the need of using ILP.

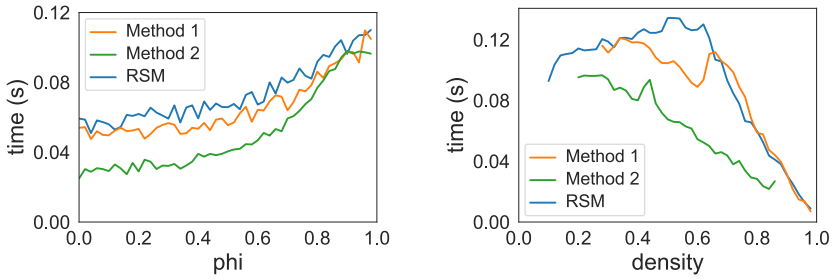


Fig. 9. Average running times of the first two phases of three-phase computation, for 10,000 RSM profiles, with 10 candidates and 100 voters, for the Borda scoring rule. Method 1 and Method 2 are from Gehrlein [10]. RSM generates challenging profiles for the computation of PW.

- We introduced new methods for generating partial orders that are of interest in their own right, most notably, the Repeated Selection Model.
- Furthermore, we produced a rich set of datasets that can serve as benchmarks in other experiments concerning incomplete preferences in computational social choice.
- We presented a number of algorithmic techniques for computing the necessary winners and the possible winners for positional scoring rules in the presence of incomplete preferences. We demonstrated that our techniques scale well in a variety of settings, including settings in which computing the possible winners is an NP-hard problem.

The algorithmic techniques and the data generation methods presented here may find applications in other frameworks, including the framework introduced in [13] and studied further in [14], which aims to bring together computational social choice and databases by supporting queries about winners in elections together with relational context about candidates, voters, and candidates' positions on issues.

REFERENCES

[1] Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. 2012. Campaigns for lazy voters: Truncated ballots. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 577–584.

[2] Dorothea Baumeister and Jörg Rothe. 2012. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Inf. Process. Lett.* 112, 5 (2012), 186–190. https://doi.org/10.1016/j.ipl.2011.11.016

[3] Nadja Betzler and Britta Dorn. 2010. Towards a dichotomy for the possible winner problem in elections based on scoring rules. *J. Comput. Syst. Sci.* 76, 8 (2010), 812–836. https://doi.org/10.1016/j.jcss.2010.04.002

[4] Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. 2009. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI, 53–58.
[5] Nadja Betzler, Rolf Niedermeier, and Gerhard J. Woeginger. 2011. Unweighted coalitional manipulation under the Borda rule is NP-hard. In *22nd International Joint Conference on Artificial Intelligence*. IJCAI.
[6] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. 2016. *Handbook of Computational Social Choice*. Cambridge University Press, New York, NY.
[7] Vishal Chakraborty and Phokion G. Kolaitis. 2020. The complexity of possible winners on partial chains. *CoRR* abs/2002.12510 (2020). arXiv:2002.12510. https://arxiv.org/abs/2002.12510.
[8] Jessica Davies, George Katsirelos, Nina Narodytska, and Toby Walsh. 2011. Complexity of and algorithms for Borda manipulation. In *25th AAAI Conference on Artificial Intelligence*. AAAI.
[9] Jean-Paul Doignon, Aleksandar Pekeč, and Michel Regenwetter. 2004. The repeated insertion model for rankings: Missing link between two subset choice models. *Psychometrika* 69, 1 (2004), 33–54.
[10] William V. Gehrlein. 1986. On methods for generating random partial orders. *Oper. Res. Lett.* 5, 6 (1986), 285–291.
[11] Optimisation Gurobi. 2019. *Gurobi Optimizer Reference Manual*. Gurobi Optimization LLC, Beaverton, OR. http://www.gurobi.com/.
[12] Batya Kenig. 2019. The complexity of the possible winner problem with partitioned preferences. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'19)*, Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 2051–2053. http://dl.acm.org/citation.cfm?id=3332007.
[13] Benny Kimelfeld, Phokion G. Kolaitis, and Julia Stoyanovich. 2018. Computational social choice meets databases. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press, IJCAI, 317–323.
[14] Benny Kimelfeld, Phokion G. Kolaitis, and Muhammad Tibi. 2019. Query evaluation in election databases. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19)*. ACM, 32–46.
[15] Kathrin Konczak and Jérôme Lang. 2005. Voting procedures with incomplete preferences. In *Proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, Vol. 20. IJCAI.
[16] C. L. Mallows. 1957. Non-null ranking models. I. *Biometrika* 44, 1–2 (June 1, 1957), 114–130.
[17] Sergey Polyakovskiy, Rudolf Berghammer, and Frank Neumann. 2016. Solving hard control problems in voting systems via integer programming. *Eur. J. Oper. Res.* 250, 1 (2016), 204–213.
[18] Shini Renjith, A. Sreekumar, and M. Jathavedan. 2018. Evaluation of partitioning clustering algorithms for processing social media data in tourism domain. In *2018 IEEE Recent Advances in Intelligent Computational Systems (RAICS'18)*. IEEE, 127–131.
[19] Julia Stoyanovich, Lovro Ilijasic, and Haoyue Ping. 2016. Workload-driven learning of mallows mixtures with pairwise preference data. In *Proceedings of the 19th International Workshop on Web and Databases*. ACM, 8. https://doi.org/10.1145/2932194.2932202
[20] Lirong Xia and Vincent Conitzer. 2011. Determining possible and necessary winners given partial orders. *J. Artif. Intell. Res.* 41 (2011), 25–67. https://doi.org/10.1613/jair.3186
[21] Yongjie Yang. 2014. Election attacks with few candidates. In *ECAI (Frontiers in Artificial Intelligence and Applications)*, Vol. 263. IOS Press, 1131–1132.

Received May 2020; revised November 2020; accepted March 2021