High-Multiplicity Election Problems

Zack Fitzsimmons
College of the Holy Cross
Worcester, MA
zfitzsim@holycross.edu

Edith Hemaspaandra
Rochester Institute of Technology
Rochester, NY
eh@cs.rit.edu

ABSTRACT

The computational study of elections generally assumes that the preferences of the electorate come in as a list of votes. Depending on the context, it may be much more natural to represent the list succinctly, as the distinct votes of the electorate and their counts, i.e., high-multiplicity representation. We consider how this representation affects the complexity of election problems. High-multiplicity representation may be exponentially smaller than standard representation, and so many polynomial-time algorithms for election problems in standard representation become exponential-time. Surprisingly, for polynomial-time election problems, we are often able to either adapt the same approach or provide new algorithms to show that these problems remain polynomial-time in the high-multiplicity case; this is in sharp contrast to the case where each voter has a weight, where the complexity usually increases. In the process we explore the relationship between high-multiplicity scheduling and manipulation of high-multiplicity elections. And we show that for any fixed set of job lengths, high-multiplicity scheduling on uniform parallel machines is in P, which was previously known for only two job lengths. We did not find any natural case where a polynomial-time election problem does not remain in P when moving to high-multiplicity representation. However, we found one natural NP-hard election problem where the complexity does increase, namely winner determination for Kemeny elections.

ACM Reference Format:
Zack Fitzsimmons and Edith Hemaspaandra. 2018. High-Multiplicity Election Problems. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Elections are an important and widely used tool for determining an outcome given the preferences of several agents (see, e.g., [6]). In most of the computational studies on elections, the preferences of the voters are represented as a list of votes. Though this may be a reasonable representation for paper ballots in political elections, in artificial intelligence applications a more succinct representation where the preferences of the electorate are represented as a list of distinct votes and their counts may be more natural. For example, this representation is used by the online preference repository PrefLib for election data [32]. Following terminology from scheduling introduced by Hochbaum and Shamir [26] we refer to this as high-multiplicity representation.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

We consider how high-multiplicity representation can affect the complexity of different election problems, and contrast this with the case of weighted voters.

High-multiplicity representation may be exponentially smaller than standard representation, and so many polynomial-time algorithms for election problems in standard representation become exponential-time. Surprisingly, we find that the complexity of polynomial-time election problems generally remains in P. We explain this phenomenon by showing that many common proof techniques that show that election problems are in P can be adapted to the high-multiplicity case (see Section 3).

We also explore the relationship between the well-studied problem of high-multiplicity scheduling and high-multiplicity election manipulation (see Section 4). In the process, we show that for any fixed set of job lengths, high-multiplicity scheduling on uniform parallel machines (this is the case where different machines can have different speeds) is in P (Theorem 4.2). Previous work showed that for two job lengths, high-multiplicity scheduling on uniform parallel machines is in P [33], and very recent work shows that for any fixed number of job lengths, high-multiplicity scheduling on *identical* parallel machines (basically bin packing) is in P [16].

In general, we find that the complexity of polynomial-time election problems in standard representation does not increase when moving to the high-multiplicity case. This is in line with related work that considers high-multiplicity representation of elections (there simply called succinct elections), which did not find a case where the complexity increases [10, 12, 13, 19]. And this general behavior is also found in the literature on high-multiplicity scheduling. For example, with respect to scheduling, Clifford and Posner [7] states: "...we do not know of a problem that can be solved in polynomial time, under standard encoding, yet is NP-complete under HM encoding."

However, this does not mean that it is not possible for the complexity to increase. For example, consider the following election system ExactlyHalfApproval. Every voter votes by an approval vector (i.e., states an approval or disapproval for each candidate), and a candidate is a winner if they receive exactly half of all approvals. Though simple, this election system is not particularly natural. It has been chosen to show that an increase in complexity is possible. Consider the problem of constructive control by adding candidates (CCAC) for election system ExactlyHalfApproval. Given an election in high-multiplicity representation, a set of unregistered candidates, and a preferred candidate p, it is NP-complete to determine whether p can be made a winner of the election using system ExactlyHalfApproval by adding unregistered candidates to the election. This is because this problem is basically subset sum, which is in P for unary numbers by dynamic programming and is NP-complete for binary numbers [15]. (See Theorem 6.1 for a

straightforward proof of this example. Also, we mention in passing that ExactlyHalfApproval-Weighted-CCAC is also NP-complete.)

We did find a natural case where the complexity increases, but for an NP-hard election problem. We find that determining the winner of a Kemeny election, which is well-known to be Θ_2^p-complete for standard representation [25], is Δ_2^p-complete for high-multiplicity representation. This solves an open problem from the previous work by Hemaspaandra, Spakowski, and Vogel [25]. (See Section 6.)

2 PRELIMINARIES

An *election* (in *standard representation*) is defined as a finite set of candidates C, and a list of voters V, where each voter $v \in V$ has a preference order over the candidates. In high-multiplicity representation, V is not a list of voters, but instead a list of distinct votes v (preference orders) and their positive integer counts $\kappa(v)$. To see how these two definitions compare, consider the election that in standard representation consists of $C = \{a, b, c\}$ and the following four voters: $(a > b > c)$, $(a > b > c)$, $(a > b > c)$, and $(b > c > a)$. This list of four voters in high-multiplicity representation is: $3 \times (a > b > c)$ and $1 \times (b > c > a)$.

We will compare complexity results between high-multiplicity and weighted election problems. In a *weighted election*, V is still a list of voters, but now each $v \in V$ has an associated positive integer weight $\omega(v)$, and can be thought of as a coalition of $\omega(v)$ voters all voting the same. It is important to note that weights are indivisible, while the counts in high-multiplicity representation are not.[1] We assume that the preference order of each voter is a total order, i.e., he or she strictly ranks each candidate from most to least preferred.

An election system $\mathcal{E}$ maps an election to a set of winners, where the winners can be any subset of the candidate set. (This is known as the nonunique winner model.) The problem $\mathcal{E}$-Winner is defined below.

Name: $\mathcal{E}$-Winner
Given: An election (C, V) and a candidate $p \in C$.
Question: Is p a winner of (C, V) using election system $\mathcal{E}$?

In the high-multiplicity case, $\mathcal{E}$-High-Multiplicity-Winner, we use high-multiplicity representation for V, and in the weighted case, $\mathcal{E}$-Weighted-Winner, each voter has a corresponding positive integer weight.

A scoring vector $\langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$, $\alpha_i \geq \alpha_{i+1}$, defines an election system over m candidates. Each candidate receives α_i points for each vote where they are ranked ith, and the candidate(s) with the highest score win.

A *pure scoring rule* defines a family of scoring vectors where the m-candidate scoring vector can be computed in polynomial time in m, and the $(m+1)$-candidate scoring vector can be obtained from the m-candidate scoring vector by adding a single coefficient [4].

2.1 Manipulative Actions

We examine the complexity of the following manipulative actions on elections. For each problem, we present the definition for the standard representation, and describe how the high-multiplicity and weighted cases differ.

Manipulation is the most widely studied manipulative action on elections. The computational study of manipulation was introduced by Bartholdi, Tovey, and Trick [2], and was later extended for the coalitional, destructive, and weighted cases by Conitzer, Sandholm, and Lang [8]. We present the definition of constructive unweighted coalitional manipulation (CUCM) below.

Name: $\mathcal{E}$-CUCM
Given: A set of candidates C, a list of nonmanipulative voters V_1, a list of k manipulators V_2,[2] and a preferred candidate $p \in C$.
Question: Is there a way to set the preferences of the manipulators so that p is a winner of the election $(C, V_1 \cup V_2)$ using election system $\mathcal{E}$?

The weighted case (CWCM) is essentially the same as above, except each of the voters (both the nonmanipulators and the manipulators) have an associated positive integer weight. In the corresponding high-multiplicity case, we use high-multiplicity representation for the nonmanipulators as well as for the manipulators, which are represented as k encoded in binary.

Electoral control denotes the family of manipulative actions that consider an agent with control over the structure of the election, called the election chair, who wants to ensure his or her preferred outcome [3]. Below we define constructive control by adding voters (CCAV), which is a very natural case of control and can be thought of as modeling get-out-the-vote drives.

Name: $\mathcal{E}$-CCAV
Given: A set of candidates C, a list of registered voters V, a list of unregistered voters U, an add limit $k \in \mathbb{N}$, and a preferred candidate $p \in C$.
Question: Does there exist a list U' of unregistered voters $U' \subseteq U$ such that U' consists of at most k unregistered voters and p is a winner of $(C, V \cup U')$ using election system $\mathcal{E}$?

In the standard model of weighted voter control the parameter k denotes the number of weighted voters the chair can add/delete [11]. In the high-multiplicity case, the only change from the $\mathcal{E}$-CCAV definition above is that we use high-multiplicity representation for both lists of voters.

2.2 Computational Complexity

We assume that the reader is familiar with the complexity classes P and NP. We also have results concerning the complexity classes Θ_2^p and Δ_2^p. Δ_2^p denotes the class of problems solvable in P with access to an NP oracle. Θ_2^p is a subset of Δ_2^p, where the P-machine can ask a log number of queries to an NP oracle (which is equivalent to the class of problems solvable by a P-machine that can ask one round of parallel queries to an NP oracle [17]).

Most of our polynomial-time results for high-multiplicity problems modify the proof of the problem in standard representation, but some results will use the well-known result due to Lenstra, which shows that even though solving an integer linear program is NP-complete in the general case [5, 27], it is in P when the number

[1] Xia, Conitzer, and Procaccia [35] have a notion of "divisible" weights, but they allow noninteger divisions and so this is very different from high-multiplicity. It is interesting that their paper reduces manipulation for this notion to a scheduling problem, though a different scheduling problem than what we use.

[2] This can also be thought of as specifying the number of manipulators, k, in unary.

of variables is fixed [29]. Informally, an integer linear program is a system of linear inequalities with integer variables and coefficients.

3 ADAPTING APPROACHES

The general theme of this paper is that even though we would expect the complexity of election problems to increase (with respect to the length of the input) when using high-multiplicity representation, we find that in general such increases do not occur. In this section we will discuss several common approaches used to show that election problems are in P (in standard representation), and describe how they can be adapted for high-multiplicity representation, sometimes straightforwardly and sometimes in a more complicated way.

In particular, we show how greedy approaches, limited brute-forcing, network flow, and edge matching/cover techniques can be adapted. To showcase these adaptations, we will show that the dichotomy result for constructive control by adding voters (CCAV) for pure scoring rules [21] holds for the high-multiplicity case.

Assuming P ≠ NP, the following pure scoring rules are asymptotically (i.e., for a large enough number of candidates) the only cases where CCAV is in P, both for standard and high-multiplicity representation.

- $\langle \alpha, \beta, 0, \ldots, 0 \rangle$, where $\alpha > \beta$.
- t-Approval: $\langle 1^t, 0, \ldots, 0 \rangle$, where $t \leq 3$.
- t-Veto: $\langle 1, \ldots, 1, 0^t \rangle$, where $t \leq 2$.
- $\langle 2, 1, \ldots, 1, 0 \rangle$.

In contrast, all weighted cases are NP-complete, except triviality (i.e., a scoring rule where every candidate scores the same), 1-approval (plurality), 2-approval, and 1-veto [11, 31].

The following is a case where we need to adapt a greedy approach as well as to handle limited brute-forcing.

Theorem 3.1. *For $\alpha > \beta \geq 0$, $\langle \alpha, \beta, 0, \ldots, 0 \rangle$-CCAV is in* P *for elections in high-multiplicity representation.*

Proof. We follow the proof of the claim from [21], which is found in its corresponding technical report [22], and show how to adapt this to the high-multiplicity case.

The proof shows that there is a constant ℓ such that it is never better to add ℓ votes that put p second than it is to add ℓ different votes that put p first (assuming that votes that rank the first two candidates identically also rank all other candidates identically). Let V_1 be the set of unregistered voters that put p first and let V_2 be the set of unregistered voters that put p second.

So, if p can be made a winner, p can be made a winner by adding at most ℓ voters from V_2 or in a way that does not leave ℓ different votes in V_1 unused.

In the first case, we can brute-force over all sets of V_2 voters of size $\leq \ell$. Add these voters. We are then left with the problem of checking whether we can add at most k' V_1 voters to an election to make p a winner, where k' is k minus the number of V_2 voters added. In Hemaspaandra, Hemaspaandra, and Schnoor [21], this is done by brute-forcing over every j in $\{0, \ldots, k'\}$ and checking whether we can add j voters from V_1 to make p a winner. If we know j, we know what the score of p after control will be, and so the last part can then be done greedily, by for each candidate adding as many voters that put a second as possible one voter at the time (until we have added j voters total).

We have two problems in the high-multiplicity case. The first is that we cannot in polynomial time brute-force over all possible values of j, since k' is not polynomially bounded. The second problem is that we don't have the time to add voters one at the time. The second problem can of course easily be fixed by, for each candidate a, adding as many voters with a in second place as possible all at once: If s_a is the initial score of a and fs_p is the final score of p (note that $fs_p = s_p + j\alpha$), we can add at most $\lfloor (fs_p - s_a)/\beta \rfloor$ voters with a second.

To handle the first problem, note that we can formulate the problem as an integer linear program with one variable (j), namely, there is a j, $0 \leq j \leq k'$ such that

$$\sum_{a \in C - \{p\}} \min(\lfloor (s_p + j\alpha - s_a)/\beta \rfloor, v_a) \geq j,$$

where v_a is the number of voters in V_1 that rank a second.[3] This integer linear program can be solved in polynomial time by Lenstra [29].

It remains to show that the second case, where p can be made a winner in a way that does not leave ℓ different votes in V_1 unused can be determined in polynomial time in the high-multiplicity case.

For the standard case, in Hemaspaandra, Hemaspaandra, and Schnoor [21] this case is handled by brute-forcing over all sets S of at most $\ell - 1$ candidates who are ranked second by unused V_1 voters, and then brute-forcing over all possible sets of unused V_1 voters consistent with S. We can describe a set of voters consistent with S as a function $u : S \to \{1, \ldots, \|V_1\|\}$ such that $u(c)$ is the number of unused V_1 voters that rank c second. We can brute-force over all such functions u in polynomial time. This way, we loop over all possible sets of unused V_1 voters, and so we also loop over all possible sets of added V_1 voters. After adding the V_1 voters, we need to see if p can be made a winner by adding V_2 voters. This can be done in a similar way as adding the V_1 voters in the first case above.

In the high-multiplicity case, we can still loop over all sets S of at most $\ell - 1$ candidates. We can not brute-force all functions u, but this will turn again into an integer linear program, this time with $\ell - 1$ variables (for the u values) and one more variable to handle the V_2 voters similarly to how we handled the V_1 voters in the first case above. This integer linear program can be solved in polynomial time by Lenstra [29]. ❑

We now show how edge matching/cover techniques can be adapted.

Theorem 3.2. *CCAV is in* P *for t-approval when $t \leq 3$ and for t-veto when $t \leq 2$ for high-multiplicity elections.*

Proof. The results for 1-approval, 2-approval, and 1-veto follow via simple greedy algorithms that can easily be adapted to work in the high-multiplicity case.

3-approval-CCAV was shown to be in P by Lin [31] using a reduction to Simple b-Edge Matching for Multigraphs (see [34]). The essence of the reduction is that every unregistered voter of the form $(\{p, x, y\} > \cdots)$ (by which we mean a voter who gives a

[3] Notice that the ILP above checks that if we add the maximum number of voters such that the score of each of the non-p candidates is at most $s_p + j\alpha$, then we have added at least j voters. This implies that we can add j voters in such a way that p is still a winner.

point to p, x, and y) corresponds to an edge (x, y) in the constructed multigraph. For every candidate $c \neq p$, c is a vertex in the graph and $b(c)$ is the final score of p minus the initial score of c (i.e., the number of points that we can add to c while keeping c's score less than or equal to p's score). In the high-multiplicity case, we would have too many edges in the graph. However, Capacitated b-Edge Matching (for graphs where edges have integer capacities) is also in P (see [34]). So, we simply set the capacity of edge (x, y) equal to the number of unregistered voters of the form $(\{p, x, y\} > \cdots)$, and we let $b(c)$ be the final score of p minus the initial score of c as previously.

Similarly, 2-veto-CCAV was shown to be in P by reduction to Simple b-Edge Cover for Multigraphs. In that reduction, every unregistered voter of the form $(\cdots > \{x, y\})$ corresponds to an edge (x, y) in the constructed multigraph. Again, we can modify this construction to a reduction to Capacitated b-Edge Cover, which is also in P, by letting the capacity of edge (x, y) be the number of unregistered voters of the form $(\cdots > \{x, y\})$. ❑

Another common technique to show that election problems are in P is network flow.

Theorem 3.3. *$\langle 2, 1, \ldots, 1, 0\rangle$-CCAV is in* P *for high-multiplicity elections.*

Proof. The essence of the standard representation proof is to (after some easy preprocessing) build a min-cost flow network. The capacities of the edges are the scores of the candidates from the registered voters and the multiplicities of the votes of the unregistered voters. In the high-multiplicity case, we can use the exact same network. The capacities are now binary integers, but min-cost network flow is still in P in that case. ❑

Dynamic Programming There is one other common technique that is used to prove that election problems are in P. This is dynamic programming. As alluded to in the introduction, dynamic programming approaches do not generalize to the high-multiplicity case. Let's for example look at the result by Hemaspaandra and Schnoor [24] that shows that for all pure scoring rules with a constant number of different coefficients, manipulation is in P. This is shown by dynamic programming. And this algorithm is not in P for the high-multiplicity case, not even for very restricted versions of this problem. In the next section we consider a different approach to this problem using scheduling.

4 MANIPULATION AS SCHEDULING

We now turn to the problem of manipulation for high-multiplicity elections for pure scoring rules. What does our election manipulation problem have to do with scheduling? As an example, consider scoring rule $\langle 4, 3, 3, 2, 0, \ldots, 0\rangle$ and let $\{p, c_1, \ldots, c_m\}$ be the set of candidates. Let s_c be the score of candidate c from the nonmanipulators, and let k be the number of manipulators. Note that we can assume that all manipulators rank p first, so that p's final score is $s_p + 4k$. We need to see if we can set the manipulators such that every candidate $c \neq p$ scores at most $s_p + 4k$. As also pointed out in Bachrach et al. [1] for the case without nonmanipulators, we can view this as the following scheduling problem: We have $2k$ jobs of length 3 and k jobs of length 2, we have m machines, machine i corresponds to candidate c_i, and machine i has deadline $s_p + 4k - s_{c_i}$. It is easy to see that if p can be made a winner, then we can schedule the jobs such that every machine meets its deadline. However, we note that the converse does not hold: We need to make sure that every machine has at most k jobs scheduled on it. We also need to be careful that every voter ranks each candidate exactly once. The construction from Hemaspaandra and Schnoor [23] shows that there is a successful manipulation if and only if there is a successful schedule such that each machine has at most k jobs scheduled on it. Note that this also gives the obvious equivalence for the corresponding high-multiplicity versions of these problems.

More generally, we can state the following equivalence, which is implicit in the work by Hemaspaandra and Schnoor [23], which shows that manipulation is in P for all pure scoring rules with a constant number of different coefficients in standard representation.

Theorem 4.1. *Let $\langle \alpha_0, \alpha_1^{m_1}, \alpha_2^{m_2}, \ldots, \alpha_C^{m_C}, 0^{m_{C+1}}\rangle$ be a scoring rule such that $\alpha_0 \geq \alpha_1 > \alpha_2 > \cdots > \alpha_C > 0$ and $m_1, \ldots, m_{C+1}$ are positive integers such that $m_1 + \cdots + m_{C+1} = m$. Let the set of candidates be $\{p, c_1, \ldots, c_m\}$ and let there be k manipulators.*

Then p can be made a winner if and only if for $1 \leq j \leq C$, we can schedule km_j jobs of length α_j on m machines such that the ith machine has deadline $s_p + \alpha_0 k - s_{c_i}$ ***with the additional restriction that we schedule at most k jobs per machine****.*

Since our problem is closely related to scheduling and since high-multiplicity scheduling has been well-studied, it makes sense to see what is known there, so that we can see how easy or hard these problems are and so that we can try to adapt the results.

First some terminology. It is easy to see that scheduling with machine-dependent deadlines is equivalent to the well-known scheduling problem called scheduling on uniform parallel machines (where different machines can have different speeds) [33]. In the high-multiplicity version of the scheduling with machine-dependent deadlines problem, the input is given as $\ell_1, \ldots, \ell_C, n_1, \ldots, n_C, D_1, \ldots, D_m$, meaning that there are n_i jobs of length ℓ_i, and m machines with deadlines $D_1, \ldots, D_m$. If all deadlines are the same, the problem is equivalent to scheduling on identical parallel machines (this problem is also known as bin packing and as the cutting-stock problem). In the high-multiplicity version of that problem, the input is given as $\ell_1, \ldots, \ell_C, n_1, \ldots, n_C, m, D$ (note that this makes the input even more succinct, and so a polynomial-time algorithm for scheduling with machine-dependent deadlines does not necessarily imply a polynomial-time algorithm for the same problem where all machines have the same deadline).

In many contexts, the number of job lengths or item types is small and so high-multiplicity scheduling for a constant number of item types is an important problem. Surprisingly, this problem was open for a long time. Only in 2001 was this determined to be in P for two job lengths, both in the case where all machines have the same deadline as well as the case with machine-dependent deadlines [33]. The case for any constant number of job lengths was only very recently shown to be in P for the case where all machines have the same deadline [16]. However, the algorithm from [16] does not give a polynomial-time result for the case of machine-dependent deadlines.

Given all this, do we have any chance of solving our high-multiplicity manipulation problems? There are some glimmers of

hope. The first one is that the restriction to having at most k jobs per machine could make the problem easier (it could also make it harder of course). Secondly, we know that the number of jobs of each length is a multiple of k (this could make the problem simpler). Finally, we can take $\ell_1, \ldots, \ell_C$ to be constant, since for a pure scoring rule with a constant number of different coefficients, there are only a fixed number of coefficients that occur for any number of candidates, and by Theorem 4.1, the occurring coefficients are exactly the occurring job lengths. We will first show that for every fixed $C, \ell_1, \ldots, \ell_C$, high-multiplicity scheduling with machine-dependent deadlines (and thus also high-multiplicity scheduling on uniform parallel machines) is in P. Note that this is a very natural problem, since in many contexts, the set of job lengths or item types is fixed. So this is a very interesting result in its own right.

Theorem 4.2. *For any fixed set of job lengths, high-multiplicity scheduling on uniform parallel machines is in* P.

Proof. Let C be the number of job lengths and let $\ell_1, \ldots, \ell_C$ be these job lengths. As mentioned previously, high-multiplicity scheduling on uniform parallel machines is equivalent to high-multiplicity scheduling on machines with machine-dependent deadlines [33], and this is the problem that we will show to be in P.

Given $n_1, \ldots, n_C$, where n_i is the number of jobs of length ℓ_i, and $D_1, \ldots, D_m$, where D_i is the deadline for machine i, we can solve our scheduling problem in polynomial time, and pretty easily at that.

Example Case. We will first explain the core argument by looking at $C = 2$, $\ell_1 = 2$, and $\ell_2 = 3$ and after that we will explain how the argument generalizes. So, we have n_1 jobs of length 2, n_2 jobs of length 3, and m machines with deadlines $D_1, \ldots, D_m$. And the question is whether there exist nonnegative integers x_i (the number of jobs of length 2 scheduled on machine i) and y_i (the number of jobs of length 3 scheduled on machine i) such that $2x_i + 3y_i \leq D_i$, $\sum_{i=1}^{m} x_i = n_1$, and $\sum_{i=1}^{m} y_i = n_2$.

Consider the ith machine. We can group the jobs as "sixes," by which we mean 3 jobs of length 2 or 2 jobs of length 3, and "the leftovers," which consist of 0, 1, or 2 jobs of length 2 and 0 or 1 jobs of length 3. Note that the sum of the leftovers is at most 7.

Let D_i' be the largest integer that is $\leq D_i - 7$ and divisible by 6. Then D_i' time can be scheduled with just sixes and no leftovers. And so we can schedule successfully if and only if there exist a_1 jobs of length 2 and a_2 jobs of length 3 that can be scheduled on m machines with deadlines $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$ such that the remaining $(n_1 - a_1)$ jobs of length 2 and the remaining $(n_2 - a_2)$ jobs of length 3 can be scheduled on m machines with deadlines $D_1', \ldots, D_m'$ in sixes, i.e., as groups of 3 jobs of length 2 and groups of 2 jobs of length 3.

Why is this in P? Note that $D_i - D_i' \leq 12$, since D_i' is the largest integer that is $\leq D_i - 7$ and divisible by 6, and so $a_1 \leq 6m$ and $a_2 \leq 4m$, and we can use dynamic programming to compute all values of a_1 and a_2 such that a_1 jobs of length 2 and a_2 jobs of length 3 fit into $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$. Finally, check that $n_1 - a_1$ is divisible by 3, that $n_2 - a_2$ is divisible by 2, and that $(n_1 - a_1)/3 + (n_2 - a_2)/2 \leq (D_1' + \cdots + D_m')/6$.

This takes time $O(m^3 \log n)$, where n is the number of jobs. We can bring this down to $O(m^2 \log m \log n)$, by using Leung's approach and Leung's $O(n^2 \log m)$ dynamic programming algorithm [30].

General Case. It is easy to see how the approach for the example case generalizes to the case for each fixed set of job lengths $\ell_1, \ldots, \ell_C$. Let $\ell = lcm(\ell_1, \ldots, \ell_C)$. Our input is $n_1, \ldots, n_C, D_1, \ldots, D_m$. Consider the ith machine. We can group the jobs as "ℓs," by which we mean ℓ/ℓ_j jobs of length ℓ_j for some j, and "the leftovers," which consist of $< \ell/\ell_j$ jobs of length ℓ_j for each j. Note that the sum of the leftovers is at most $\sum_{j=1}^{C}(\ell/\ell_j - 1)$.

For each i, let D_i' be the largest integer that is $\leq \sum_{j=1}^{C}(\ell/\ell_j - 1)$ and divisible by ℓ. Then D_i' time can be scheduled with just ℓs and no leftovers. And so we can schedule successfully if and only if there exist a_j jobs of length ℓ_j, for $1 \leq j \leq C$, that can be scheduled on m machines with deadlines $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$ such that all remaining $(n_j - a_j)$ jobs of length ℓ_j, for $1 \leq j \leq C$, can be scheduled on m machines with deadlines $D_1', \ldots, D_m'$ in ℓs, i.e., as groups of ℓ/ℓ_j jobs of length ℓ_j.

Why is this in P? Note that $D_i - D_i' \leq \sum_{j=1}^{C}(\ell/\ell_j - 1) + \ell - 1$, since D_i' is the largest integer that is $\leq \sum_{j=1}^{C}(\ell/\ell_j - 1)$ and divisible by ℓ. Since C and $\ell_1, \ldots, \ell_C$ are fixed, $a_j = O(m)$ and we can use dynamic programming to compute all values of $a_1, \ldots, a_C$ such that a_j jobs of length ℓ_j fit into $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$. Finally, check that $n_j - a_j$ is divisible by ℓ_j and that $\sum_{j=1}^{C}(n_j - a_j)/\ell_j \leq (D_1' + \cdots + D_m')/\ell$.

Note that since the ℓ_i's and C are constant the running time provided for the example case also holds for the general case. ❑

Note that the above does not give a polynomial-time algorithm if only C is fixed and $\ell_1, \ldots, \ell_C$ are given as part of the input. And so the above does not imply the result that for two job lengths, high-multiplicity scheduling with machine-dependent deadlines is in P from McCormick, Smallwood, and Spieksma [33].

However, as mentioned previously, the case where $\ell_1, \ldots, \ell_C$ are fixed is very natural, and so this is an interesting result for scheduling. But recall from Theorem 4.1 that to solve manipulation, we in addition have to ensure that there are at most k jobs scheduled on each machine. As mentioned earlier, such a restriction can potentially make the complexity harder (as well as easier).

What are the problems with adding the restriction that we have at most k jobs per machine in the argument of the proof of Theorem 4.2? Let's look at the $C = 2, \ell_1 = 2, \ell_2 = 3$ case. In the case without the restriction that we have at most k jobs per machine, all deadlines with the same value mod 6 are treated the same, and every "six" is treated the same (no matter what machine it's on). This is no longer the case if we also require that every machine handles at most k jobs, since it then matters if you schedule three jobs of length 2 or two jobs of length 3 in a six.

However, we can show a number of high-multiplicity manipulation cases to be in P using scheduling, which is in contrast to the weighted case where only triviality and plurality are in P. Note that the theorem below does not cover all of the polynomial-time cases of manipulation for pure scoring rules in standard representation, since for example, we do not cover $\langle 2^{m/3}, 1^{m/3}, 0^{m/3} \rangle$ (though cases with at most two different coefficients can be handled easily).

Theorem 4.3. *Let $\alpha_0 \geq \alpha_1 > \alpha_2 > \cdots > \alpha_C > 0$. High-multiplicity manipulation for pure scoring rule $\langle \alpha_0, \alpha_1^{m_1}, \alpha_2^{m_2}, \ldots, \alpha_C^{m_C}, 0^{m-(m_1+\cdots+m_C)} \rangle$ is in* P.

Proof. By Theorem 4.1, it suffices to show that the following problem is in P: Given $k, D_1, \ldots, D_m$, can we schedule km_j jobs of length α_j on m machines with deadlines $D_1, \ldots, D_m$ such that every machine meets its deadline and such that there are at most k jobs scheduled on each machine.

Example Case. We first explain the core argument for $C = 2$, $\ell_1 = 2$, $\ell_2 = 3$, $m_1 = 1$, $m_2 = 1$. So, we are given $k, D_1, \ldots, D_m$, and we are asking whether we can we schedule k jobs of length 2 and k jobs of length 3 on m machines with deadlines $D_1, \ldots, D_m$ such that every machine meets its deadline and such that there are at most k jobs scheduled on each machine. That is, we are asking whether there exist nonnegative integers x_i (the number of jobs of length 2 scheduled on machine i) and y_i (the number of jobs of length 3 scheduled on machine i) such that $2x_i + 3y_i \leq D_i$, $\sum_{i=1}^m x_i = k$, $\sum_{i=1}^m y_i = k$, and $x_i + y_i \leq k$.

Note that we can assume that $D_i \leq 3k$. Also note that if $D_i \leq 2k$, our new requirement that machine i has at most k jobs, i.e., $x_i + y_i \leq k$, follows from the old requirement that $2x_i + 3y_i \leq D_i$.

If $D_i \leq 2k$ for all i, then we can solve our problem by using the algorithm from Theorem 4.2. If there are more than two machines with deadline $> 2k$, we can schedule k jobs of length 2 on machine 1, $\lceil k/2 \rceil$ jobs of length 3 on machine 2, and $\lfloor k/2 \rfloor$ jobs of length 3 on machine 3.

It remains therefore to look at the case where there are one or two machines with deadline $> 2k$. Without loss of generality, assume that the deadlines are in nonincreasing order and that we have at least two machines. Then our case is solvable if and only if there exist x_1, y_1, x_2, y_2 such that $2x_1 + 3y_1 \leq D_1$, $2x_2 + 3y_2 \leq D_2$, $x_1 + y_1 \leq k$, and $x_2 + y_2 \leq k$, such that $k - x_1 - x_2$ jobs of length 2 and $k - y_1 - y_2$ jobs of length 3 can be scheduled on $m - 2$ machines with deadlines $D_3, \ldots, D_m$ (since all these deadlines are $\leq 2k$, the requirement that we schedule at most k jobs on each machine will be automatically satisfied). We would like to use the algorithm from Theorem 4.2, but we would need to run that algorithm for all values of x_1, y_1, x_2, y_2, which is not polynomial. The solution is to rephrase the algorithm from Theorem 4.2 as an integer linear program with a fixed number of variables (see Lemma 4.4 below), which we then combine with the equations involving x_1, y_1, x_2, y_2 above. This ILP can be solved in polynomial time by Lenstra [29].

General Case. Now let's consider how to generalize the approach used in the example case to the general case. Note that we can assume that $D_i \leq \alpha_1 k$. Also note that if $D_i \leq \alpha_C k$, our new requirement that machine i has at most k jobs follows from the old requirement that D_i meets its deadline.

If there are at least $\alpha_1(m_1 + \cdots + m_C)$ machines with deadline $> \alpha_C k \geq k$, then we can always schedule $\lceil k/\alpha_1 \rceil$ jobs on each such machine.

Let $\widehat{m} = \alpha_1(m_1 + \cdots + m_C) - 1$. Note that $\widehat{m}$ is a constant. It remains therefore to look at the case where there are at most $\widehat{m}$ machines with deadline $> k$. Without loss of generality, assume that the deadlines are in nonincreasing order and that we have at least $\widehat{m}$ machines. Our case is solvable if and only if there exist $x_{i,j}$ for $1 \leq i \leq \widehat{m}$ and $1 \leq j \leq C$, the number of jobs scheduled on machine i of length ℓ_j, such that $\sum_{j=1}^C \alpha_i x_{i,j} \leq D_i$, $\sum_{i=1}^m x_{i,j} = km_j$, $\sum_{j=1}^C x_{i,j} \leq k$, and such that $k - \sum_{i=1}^{\widehat{m}} x_{i,j}$ jobs of length ℓ_j can be scheduled on $m - \widehat{m}$ machines with deadlines $D_{\widehat{m}+1}, \ldots, D_m$ (since all these deadlines are $\leq k$, the requirement that we schedule at most k jobs on each machine will be automatically satisfied). Use the ILP formulation of the algorithm from Theorem 4.2 to solve this last requirement and combine this with our equations for $x_{i,j}$ for $1 \leq i \leq \widehat{m}$ and $1 \leq j \leq C$, to obtain an ILP with a fixed number of variables, which can be solved in polynomial time by Lenstra [29]. ❑

Lemma 4.4. *For any fixed set of job lengths, there is an ILP with a fixed number of variables that solves high-multiplicity scheduling on parallel machines with machine-dependent deadlines.*

Proof. Let C be the number of job lengths and let $\ell_1, \ldots, \ell_C$ be these job lengths. Given $n_1, \ldots, n_C$, where n_i is the number of jobs of length ℓ_i, and $D_1, \ldots, D_m$, where D_i is the deadline for machine i. In the proof of Theorem 4.2, we proved that this scheduling problem is in P using dynamic programming. However, for the proof of Theorem 4.3 we need a different approach. Below we present an ILP formulation with a fixed number of variables that computes if we can schedule successfully that replaces the dynamic programming. Again, to show the essence of the argument, we consider the case of $C = 2$, $\ell_1 = 2$, and $\ell_2 = 3$.

Look at a proof of Theorem 4.2 (the example case). We are already pretty close: There exist a_1 and a_2 such that a_1 jobs of length 2 and a_2 jobs of length 3 fit into $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$, $n_1 - a_1$ is divisible by 3, $n_2 - a_2$ is divisible by 2, and $(n_1 - a_1)/3 + (n_2 - a_2)/2 \leq (D_1' + \cdots + D_m')/6$.

So all that is left to do is to write "a_1 jobs of length 2 and a_2 jobs of length 3 fit into $D_1 - D_1', D_2 - D_2', \ldots, D_m - D_m'$" as an integer linear program with a fixed number of variables. Recall that $D_i - D_i' \leq 12$. For $0 \leq j \leq 12$, let d_j be the number of machines i such that $D_i' = d_j$.

Our integer linear program formulation look as follows. We introduce variables $a_{j,(r,t)}$ for $0 \leq j \leq 12$ and $2r + 3t \leq d_j$. $a_{j,(r,t)}$ stands for the number of machines i such that $D_i' = d_j$ and such that D_i' is scheduled with r 2s and t 3s. We add the following obvious equations:

$$\sum_{0 \leq j \leq 12, 2r+3t \leq d_j} a_{j,(r,t)} r = a_1$$

$$\sum_{0 \leq j \leq 12, 2r+3t \leq d_j} a_{j,(r,t)} t = a_2$$

It is easy to see that the example case generalizes to the general case, using the general case of the proof of Theorem 4.2 ❑

5 FIXED NUMBERS OF CANDIDATES

It is reasonable to assume that the number of candidates in an election may be fixed. For weighted voters many problems are hard, even when the number of candidates is fixed.

Theorem 5.1. (1) *m-candidate α-CWCM is* NP*-complete for every scoring rule α that is not plurality or triviality [18].*

(2) *m-candidate α-CCAV and m-candidate α-CCDV are each* NP*-complete for every scoring rule* $\alpha = \langle \alpha_1, \ldots, \alpha_m \rangle$*, when* $\|\{\alpha_1, \ldots, \alpha_m\}\| \geq 3$*, for weighted elections [11].*

Faliszewski, Hemaspaandra, and Hemaspaandra [10] showed that high-multiplicity manipulation for scoring rules is in P for any fixed number of candidates by describing an ILP with a fixed number of variables.

Theorem 5.2. *[10] m-candidate α-CUCM is in* P *for every scoring rule α, for elections in high-multiplicity representation.*

A similar approach can be used to show that high-multiplicity constructive control by adding/deleting voters is in P for every scoring rule for a fixed number of candidates.

Theorem 5.3. *m-candidate α-CCAV and m-candidate α-CCDV are each in* P *for every scoring rule* $\alpha = \langle \alpha_1, \ldots, \alpha_m \rangle$*, for high-multiplicity elections.*

6 NATURAL INCREASE IN COMPLEXITY

In general, we find that the complexity of polynomial-time election problems in standard representation does not increase when moving to the high-multiplicity case. However, this does not mean that such an increase is not possible. Recall the election system ExactlyHalfApproval defined in the introduction.

Theorem 6.1. *ExactlyHalfApproval-CCAC is in* P *and ExactlyHalfApproval-High-Multiplicity-CCAC is* NP*-complete.*

Proof. This problem is basically subset sum, which is in P for unary numbers and NP-complete for binary numbers [15]. To show that ExactlyHalfApproval-CCAC is in P, let $s_1, \ldots, s_m$ be the approval scores of candidates $c_1, \ldots, c_m$, let c_1 be the preferred candidate and let $c_1, \ldots, c_\ell$ be the registered candidates. We want to know if there exists a subset I of $\{\ell + 1, \ldots, m\}$ of size at most k such that

$$2s_1 = \sum_{1 \leq i \leq \ell} s_i + \sum_{i \in I} s_i.$$

This can easily be computed in polynomial time, using dynamic programming. Simply let, for all $\ell + 1 \leq i \leq m$, $0 \leq k' \leq k$, $0 \leq t \leq \sum_{1 \leq i \leq m} s_i$, $S[i, t, k'] = 1$ if and only if there exists a size-k' subset I of $\{\ell + 1, \ldots, m\}$ such that $\sum_{1 \leq i \leq \ell} s_i + \sum_{i \in I} s_i = t$. This can be done in polynomial time, since there are only polynomially many i, k', and t to consider.

When the input is given in high-multiplicity representation, there are exponentially many values for t possible. In this case, we can show that it is NP-complete. We reduce from Partition: Given $\{k_1, \ldots, k_m\}$ such that $\sum k_i = 2K$, we need to determine if a subset of these integers sums to K. We have $m + 1$ candidates, preferred candidate p and candidates $c_1, \ldots, c_m$. We have K voters that approve of only p, and k_i voters that approve of only c_i. p is the only registered voter and the addition limit is m. It is immediate that there is a subset of $k_1, \ldots, k_m$ that sums to K if and only if p can be made a winner by adding candidates. ❑

As mentioned in the introduction, this system is not particularly natural. We will now present a natural case where the complexity increases, namely winner determination for Kemeny elections.

Kemeny elections were introduced in [28]. It is the unique election system that is neutral, consistent, and Condorcet consistent [36]. A candidate p is a Kemeny winner if p is ranked first in a Kemeny consensus, i.e., a linear order $>$ over the candidates that minimizes the Kendall's tau distance to V, i.e., that minimizes $\sum_{a,b \in C, a>b} \|\{v \in V \mid b >_v a\}\|$, where $>_v$ is the preference order of voter v.

Observation 6.2. *Kemeny-Weighted-Winner is equivalent to Kemeny-High-Multiplicity-Winner.*

Notice that the above observation holds since we do not modify the votes to score the election. Kemeny-Winner was shown NP-hard in [2] and Θ_2^p-complete in [25]. Θ_2^p-hardness was shown by a chain of three reductions, ultimately showing that the Θ_2^p-complete problem Min-Card-Vertex-Cover-Compare reduces to Kemeny-Winner.

Footnote 3 in [25] points out that Kemeny-High-Multiplicity-Winner is in Δ_2^p and explicitly leaves the exact complexity of this problem as an open question. We will now show that Kemeny-High-Multiplicity-Winner (and Kemeny-Weighted-Winner) are in fact Δ_2^p-complete. We first define the following weighted version of Min-Card-Vertex-Cover-Compare.

Name: Min-Weight-Vertex-Cover-Compare
Given: Vertex-weighted graphs G and H such that $\omega(G) = \omega(H)$, where $\omega(G)$ denotes the weight of the graph.[4]
Question: Is the weight of G's minimum-weight vertex cover $\leq$ the weight of H's minimum-weight vertex cover?

Lemma 6.3. *Min-Weight-Vertex-Cover-Compare polynomial-time many-one reduces to Kemeny-High-Multiplicity-Winner.*

Proof. This follows by careful inspection and modification of the proof from [25]. In particular, since that proof consists of a chain of three reductions between Θ_2^p-complete problems, we need to define suitable Δ_2^p-complete weighted versions of the two intermediate problems and show that the weighted versions of the reductions still hold. In essence, we "lift" the constructions and proofs from Θ_2^p to Δ_2^p. This works surprisingly smoothly. ❑

It remains to show the following lemma.

Lemma 6.4. *Min-Weight-Vertex-Cover-Compare is Δ_2^p-hard.*

Proof. For a formula $\phi(x_1, \ldots, x_n)$, we view an assignment for ϕ as the n-bit string $\alpha_1 \cdots \alpha_n$ such that α_i gives the assignment for variable x_i. We identify α with the binary number (between 0 and $2^n - 1$) that it represents. [20] shows that it is Δ_2^p-hard to compare the maximal satisfying assignments of two cnf formulas. By negating the variables in the formulas, we obtain the following Δ_2^p-hard promise problem, which we will reduce to Min-Weight-Vertex-Cover-Compare to prove Lemma 6.4.

Name: MINSATASG$_\leq$
Given: Two satisfiable 3cnf formulas $\phi(x_1, \ldots, x_n)$ and $\psi(x_1, \ldots, x_n)$.
Question: Is the minimal satisfying assignment of $\phi \leq$ the minimal satisfying assignment of ψ?

[4]Note that the weight of a vertex-weighted graph is the sum of the weights of each of its vertices.

We explain the main idea behind the reduction here. Let $\phi(x_1, \ldots, x_n)$ and $\psi(x_1, \ldots, x_n)$ be two satisfiable 3cnf formulas. Without loss of generality, assume that ϕ and ψ have the same number m of clauses (simply pad).

Let $f(\phi)$ be the graph computed by the standard reduction from 3SAT to Vertex-Cover from [27]. We will not give the definition of the graph here, but list only the properties that we need, which follow immediately from the proof of [27]. Graph $f(\phi)$ consists of $2n + 3m$ vertices: a set L of $2n$ vertices corresponding to the literals in ϕ, i.e., $L = \{x_i, \overline{x_i} \mid 1 \leq i \leq n\}$ and a set of $3m$ clause vertices. Then the size of a minimum vertex cover of $f(\phi)$ is $n + 2m$ and satisfying assignments of ϕ correspond to minimum-size vertex covers in the following way:

- If W is a vertex cover of size $n + 2m$, then the set of literals $W \cap L$ corresponds to a satisfying assignment for ϕ.
- If α is a satisfying assignment for ϕ, then there is a vertex cover W of size $n + 2m$ such that $W \cap L$ corresponds to α.

Now we set the weights of the vertices so that the minimal satisfying assignment corresponds to the minimum-weight vertex cover. For each of the n vertices corresponding to variables, set $\omega(x_i) = 2^n + 2^{n-i}$ and set the weight of all other vertices to 2^n. It is not too hard to see that a minimum-weight vertex cover has size $n + 2m$ and that the weight of a size $n + 2m$ vertex cover corresponding to satisfying assignment α is $(n + 2m)2^n + \alpha$ and so α is the smallest satisfying assignment of ϕ if and only if $f(\phi)$ has a minimum-weight vertex cover of weight $(n + 2m)2^n + \alpha$.

Since ψ is also a satisfiable 3cnf formula over $x_1, \ldots, x_n$ with m clauses, it also holds α is the smallest satisfying assignment of ψ if and only if $f(\psi)$ has a minimum-weight vertex cover of weight $(n + 2m)2^n + \alpha$.

This then implies that ϕ's minimal satisfying assignment is $\leq \psi$'s minimal satisfying assignment if and only if the weight of $f(\phi)$'s minimum-weight vertex cover is less than or equal to the weight of $f(\psi)$'s minimum-weight vertex cover. Note that $\omega(f(\phi)) = \omega(f(\psi))$. This completes the reduction from MINSATASG$_{\leq}$ to Min-Weight-Vertex-Cover-Compare. ❑

This completes the proof of the main theorem of this section.

Theorem 6.5. *Kemeny-Weighted-Winner and Kemeny-High-Multiplicity-Winner are Δ_2^p-complete.*

Dwork et al. [9] show that Kemeny-Winner is already NP-hard for four voters. In fact, one can easily combine the techniques from Hemaspaandra, Spakowski, and Vogel [25] and Dwork et al. [9] to obtain the following theorem.

Theorem 6.6. *Kemeny-Winner for four voters is Θ_2^p-complete.*[5]

Do we get the same complexity jump if we look at the high-multiplicity and weighted case for four votes? Note that for the high-multiplicity case we allow many voters with the same vote, and so the Kemeny scores can be large. However, even though the scores can be large, it is easy to see that for each instance there are only a polynomial number of possible scores for each candidate, namely, if the multiplicities of the four votes are k_1, k_2, k_3, and k_4, the only possible scores are $\ell_1 k_1 + \ell_2 k_2 + \ell_3 k_3 + \ell_4 k_4$, where $0 \leq \ell_i \leq \|C\|(\|C\| - 1)/2$. For each candidate, simply query the Kemeny-Weighted-Score oracle for all these possible scores in parallel, and then determine whether p has the highest score.

Theorem 6.8. *Kemeny-Weighted-Winner and Kemeny-High-Multiplicity-Winner for four votes are Θ_2^p-complete.*

[5]The definition of Kemeny winner can naturally be extended to voters voting any binary preference relation, and we note that the same argument as for Theorem 6.6 also shows the following.

Theorem 6.7. *Kemeny-Winner for two voters who vote a partial order and Kemeny-Winner for one voter who votes a binary relation are Θ_2^p-complete.*

7 CONCLUSIONS AND FUTURE WORK

High-multiplicity representation is a very natural way to represent an election. This representation may be exponentially smaller than the standard representation and so we would expect to see an increase in complexity (with respect to the length of the input). However, we were able to either adapt the approaches used for standard representation or provide new algorithms to show that polynomial-time election problems generally remain in P. We also explored the relationship between high-multiplicity scheduling and manipulation of high-multiplicity elections, which led to a new result in scheduling.

There are several interesting directions for future work. Is it possible for a high-multiplicity election problem to be harder than the corresponding weighted problem? Can we find natural cases, for election problems as well as for scheduling problems, where the complexity increases from P to NP-hard when going to high-multiplicity representation?

ACKNOWLEDGMENTS

We thank the referees for their helpful comments and suggestions. And we thank the AAAI-17 Student Abstract referees for their helpful comments and suggestions on our preliminary work on this topic [14]. This work was supported in part by a National Science Foundation Graduate Research Fellowship under NSF grant no. DGE-1102937.

REFERENCES

[1] Y. Bachrach, O. Lev, Y. Lewenberg, and Y. Zick. 2016. Misrepresentation in District Voting. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. IJCAI/AAAI Press, 81–87.
[2] J. Bartholdi, III, C. Tovey, and M. Trick. 1989. The Computational Difficulty of Manipulating an Election. *Social Choice and Welfare* 6, 3 (1989), 227–241.
[3] J. Bartholdi, III, C. Tovey, and M. Trick. 1992. How Hard is it to Control an Election? *Mathematical and Computer Modeling* 16, 8/9 (1992), 27–40.
[4] N. Betzler and B. Dorn. 2009. Towards a Dichotomy of Finding Possible Winners in Elections Based on Scoring Rules. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag *Lecture Notes in Computer Science #5734*, 124–136.
[5] I. Borosh and L. Treybig. 1976. Bounds on Positive Integral Solutions of Linear Diophantine Equations. *Proc. Amer. Math. Soc.* 55, 2 (1976), 299–304.
[6] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia. 2016. *Handbook of Computational Social Choice*. Cambridge University Press.
[7] J. Clifford and M. Posner. 2001. Parallel machine scheduling with high multiplicity. *Math. Program.* 89, 3 (2001), 359–383.
[8] V. Conitzer, T. Sandholm, and J. Lang. 2007. When Are Elections with Few Candidates Hard to Manipulate? *J. ACM* 54, 3 (2007), Article 14.
[9] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. 2001. Rank aggregation methods for the Web. In *Proceedings of the 10th International World Wide Web Conference*. ACM Press, 613–622.
[10] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. 2009. How Hard is Bribery in Elections? *Journal of Artificial Intelligence Research* 35 (2009), 485–532.

[11] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. 2015. Weighted Electoral Control. *Journal of Artificial Intelligence Research* 52 (2015), 507–542.

[12] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2009. Llull and Copeland Voting Computationally Resist Bribery and Constructive Control. *Journal of Artificial Intelligence Research* 35 (2009), 275–341.

[13] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2011. The Shield that Never Was: Societies with Single-Peaked Preferences are More Open to Manipulation and Control. *Information and Computation* 209, 2 (2011), 89–107.

[14] Z. Fitzsimmons and E. Hemaspaandra. 2017. The Complexity of Succinct Elections. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (Student Abstract)*. AAAI Press, 4921–4922.

[15] M. Garey and D. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

[16] M. Goemans and T. Rothvoß. 2014. Polynomiality for Bin Packing with a Constant Number of Item Types. In *Proceedings of the Twenty-Fifth Annual Symposium on Discrete Algorithms*. SIAM, 830–839.

[17] L. Hemachandra. 1989. The Strong Exponential Hierarchy Collapses. *J. Comput. System Sci.* 39, 3 (1989), 299–322.

[18] E. Hemaspaandra and L. Hemaspaandra. 2007. Dichotomy for Voting Systems. *J. Comput. System Sci.* 73, 1 (2007), 73–83.

[19] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2009. Hybrid Elections Broaden Complexity-Theoretic Resistance to Control. *Mathematical Logic Quarterly* 55, 4 (2009), 397–424.

[20] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2014. The Complexity of Online Manipulation of Sequential Elections. *J. Comput. System Sci.* 80, 4 (2014), 697–710.

[21] E. Hemaspaandra, L. Hemaspaandra, and H. Schnoor. 2014. A Control Dichotomy for Pure Scoring Rules. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*. AAAI Press, 712–720.

[22] E. Hemaspaandra, L. Hemaspaandra, and H. Schnoor. 2014. *A Control Dichotomy for Pure Scoring Rules*. Technical Report arXiv:1404.4560 [cs.GT]. arXiv.org.

[23] E. Hemaspaandra and H. Schnoor. 2016. *Complexity Dichotomies for Unweighted Scoring Rules*. Technical Report arXiv:1604.05264 [cs.CC]. arXiv.org.

[24] E. Hemaspaandra and H. Schnoor. 2016. Dichotomy for Pure Scoring Rules under Manipulative Electoral Actions. In *Proceedings of the 22nd European Conference on Artificial Intelligence*. IOS Press, 1071–1079.

[25] E. Hemaspaandra, H. Spakowski, and J. Vogel. 2005. The Complexity of Kemeny Elections. *Theoretical Computer Science* 349, 3 (2005), 382–391.

[26] D. Hochbaum and R. Shamir. 1991. Strongly Polynomial Algorithms for the High Multiplicity Scheduling Problem. *Operations Research* 39, 4 (1991), 648–653.

[27] R. Karp. 1972. Reducibilities among combinatorial problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher (Eds.). 85–103.

[28] J. Kemeny. 1959. Mathematics Without Numbers. *Daedalus* 88 (1959), 577–591.

[29] H. Lenstra, Jr. 1983. Integer Programming with a Fixed Number of Variables. *Mathematics of Operations Research* 8, 4 (1983), 538–548.

[30] J. Leung. 1982. On Scheduling Independent Tasks with Restricted Execution Times. *Operations Research* 30, 1 (1982), 163–171.

[31] A. Lin. 2012. *Solving Hard Problems in Election Systems*. Ph.D. Dissertation. Rochester Institute of Technology, Rochester, NY.

[32] N. Mattei and T. Walsh. 2013. PrefLib: A Library for Preferences. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory*. 259–270.

[33] T. McCormick, S. Smallwood, and F. Spieksma. 2001. A Polynomial Algorithm for Multiprocessor Scheduling with Two Job Lengths. *Mathematics of Operations Research* 26, 1 (2001), 31–49.

[34] A. Schrijver. 2003. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag.

[35] L. Xia, V. Conitzer, and A. Procaccia. 2010. A Scheduling Approach to Coalitional Manipulation. In *Proceedings of the 11th ACM Conference on Electronic Commerce*. ACM Press, 275–284.

[36] H. Young and A. Levenglick. 1978. A Consistent Extension of Condorcet's Election Principle. *SIAM J. Appl. Math.* 35, 2 (1978), 285–300.