The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Very Hard Electoral Control Problems

Zack Fitzsimmons
College of the Holy Cross
Worcester, MA 01610

Edith Hemaspaandra
Rochester Inst. of Technology
Rochester, NY 14623

Alexander Hoover
University of Chicago
Chicago, IL 60637

David E. Narváez
Rochester Inst. of Technology
Rochester, NY 14623

Abstract

It is important to understand how the outcome of an election can be modified by an agent with control over the structure of the election. Electoral control has been studied for many election systems, but for all these systems the winner problem is in P, and so control is in NP. There are election systems, such as Kemeny, that have many desirable properties, but whose winner problems are not in NP. Thus for such systems control is not in NP, and in fact we show that it is typically complete for Σ_2^p (i.e., $\mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial hierarchy). This is a very high level of complexity. Approaches that perform quite well for solving NP problems do not necessarily work for Σ_2^p-complete problems. However, answer set programming is suited to express problems in Σ_2^p, and we present an encoding for Kemeny control.

Introduction

The study of elections often deals with trade-offs for different properties that an election system satisfies. Elections have a wide range of applications from voting in political elections to applications in artificial intelligence (see, e.g., Brandt et al. (2016)). And given the role of elections in multiagent system settings, it is important that we study the computational properties of election systems.

Attacks on the structure of an election, referred to as control, were introduced by Bartholdi, Tovey, and Trick (1992) and model natural scenarios where an agent, with control over the structure of an election, modifies the structure (e.g., by adding candidates) to ensure that their preferred candidate wins. It is important to study how these types of attacks on the structure of an election can affect the outcome and how computationally difficult it is to determine if such an attack exists.

The complexity of electoral control has been studied for many different natural elections systems and has been an important line of research in computational social choice (see, e.g., Faliszewski and Rothe (2016)). However, for all of those systems the winner problems are in P, and so the standard control problems are in NP. To go beyond what can easily be encoded for SAT solvers, we need to look at election systems with harder winner problems. But even for election systems whose winner problems are in NP, the most common control problems are still in NP.

There are election systems that have many desirable social-choice properties, but whose winner problems are not in NP (assuming $\mathrm{NP} \neq \mathrm{coNP}$). One important example is the Kemeny rule (Kemeny 1959), whose winner problem is Θ_2^p-complete (i.e., complete for parallel access to NP) (Hemaspaandra, Spakowski, and Vogel 2005) and so the complexity of the standard control problems for Kemeny are all Θ_2^p-hard and thus also not in NP. For election systems with Θ_2^p-complete winner problems, control is in Σ_2^p (i.e., $\mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial hierarchy), and we show that control is typically Σ_2^p-complete for such systems.

This is a very high level of complexity. And so a natural question to ask is how Kemeny control can be solved. We mention here that there has been a long line of work that considers ways to solve hard election problems. This includes work on computing Kemeny winners (see, e.g., (Conitzer, Davenport, and Kalagnanam 2006; Betzler et al. 2008; Ali and Meilă 2012; Betzler, Bredereck, and Niedermeier 2014)) and on solving election-attack problems (see, e.g., Rothe and Schend (2013)). The work on solving hard election-attack problems has been restricted to problems in NP, and such approaches do not work for Σ_2^p-complete problems. Answer set programming (ASP) is an approach that has been recently applied for winner determination in voting, including for systems with hard winner problems (Charwat and Pfandler 2015). ASP is suited to express Σ_2^p problems. However, encoding Σ_2^p-complete problems in ASP requires the use of more advanced techniques than encoding computationally easier problems. We present an encoding of Kemeny control using these advanced techniques and discuss other related approaches.

We make the following main contributions.

- We obtain the first natural Σ_2^p-completeness results for elections.
- We define several new natural and simple Σ_2^p-complete graph problems to prove our results, and these problems compare favorably in naturalness and simplicity to other Σ_2^p-complete problems, and so in usefulness as problems to reduce from.
- We show for the most commonly-studied election sys-

tems with Θ_2^p-complete winner problems, including the Kemeny rule, that control is typically Σ_2^p-complete.

- We build upon recent work on combining ASP with voting (Charwat and Pfandler 2015) by encoding our Σ_2^p-complete control problems using advanced ASP techniques.

Preliminaries

An election consists of a set of candidates C, and a set of voters V, where each voter has a vote that strictly ranks each candidate from most to least preferred. An election system, $\mathcal{E}$, maps an election (C, V) to a set of winners, where the winner set can be any subset of the candidate set. The winner problem for an election system, $\mathcal{E}$-Winner, is defined in the following way. Given an election (C, V) and a candidate $p \in C$, is p a winner of the election using election system $\mathcal{E}$?

We consider the most important election systems with Θ_2^p-complete winner problems: Kemeny, Young, and Dodgson.

A candidate is a Kemeny winner if it is the most-preferred candidate in a Kemeny consensus (Kemeny 1959), which is a total order "$>$" that minimizes the sum of Kendall's Tau distances (i.e., number of pairwise disagreements) with the voters in an election, i.e., minimizes $\sum_{a,b \in C, a > b} \|\{v \in V \mid b >_v a\}\|$, where $>_v$ denotes the preference of voter v.

A candidate is a Young winner if it can become a weak Condorcet winner (a candidate that beats-or-ties every other candidate pairwise) by deleting the fewest voters (Young 1988).

A candidate is a Dodgson winner if it can become a Condorcet winner (a candidate that beats every other candidate pairwise) with the fewest swaps between adjacent candidates in the voters' rankings (Dodgson 1876).

Electoral control models the actions of an agent, referred to as the election chair, who modifies the structure of the election (e.g., the voters) to ensure that their preferred candidate wins (in the constructive case) (Bartholdi, Tovey, and Trick 1992) or that their despised candidate does not win (in the destructive case) (Hemaspaandra, Hemaspaandra, and Rothe 2007).[1] (These two papers define the standard control actions.) We formally define constructive control by adding candidates (CCAC) below, which models the real-world scenario of an election chair adding spoiler candidates to an election to ensure that their preferred candidate wins.

Name: $\mathcal{E}$-CCAC
Given: A set of registered candidates C, a set of unregistered candidates D, a set of voters V having preferences over $C \cup D$, an addition limit k, and a preferred candidate $p \in C$.
Question: Does there exist a set $D' \subseteq D$ such that $\|D'\| \leq k$ and p is a winner of $(C \cup D', V)$ using election system $\mathcal{E}$?

[1]We mention here that early work that considered electoral control generally used the unique winner model whereas we allow multiple winners. This rarely makes a difference in the complexity for concrete systems.

Computational Complexity Our results concern the complexity classes Θ_2^p and Σ_2^p. The class Θ_2^p was introduced in (Papadimitriou and Zachos 1983), named in (Wagner 1990), and shown to be equivalent to $\mathrm{P}_{||}^{\mathrm{NP}}$, the class of problems that can be solved by a polynomial-time machine with parallel access to an NP oracle, in (Hemachandra 1989). $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ is the class of problems solvable by a nondeterministic polynomial-time machine with access to an NP oracle, and is the second level of the polynomial hierarchy (Meyer and Stockmeyer 1972; Stockmeyer 1976). Note that $\mathrm{NP} \cup \mathrm{coNP} \subseteq \Theta_2^p \subseteq \mathrm{P}^{\mathrm{NP}} \subseteq \Sigma_2^p$.

Complexity Results

In this section we show that control problems for Kemeny, Young, and Dodgson elections are typically Σ_2^p-complete.

Observation 1 *For an election system $\mathcal{E}$, the complexity of each standard control action is in* $\mathrm{NP}^{\mathcal{E}\text{-Winner}}$.

It is easy to see that the above observation holds. For a given election, guess the control action of the chair (e.g., the set of candidates to be added) and then use the oracle to check that the preferred candidate is a winner (or that the despised candidate is not a winner). In the case of partition control, which will not be discussed further in this paper, the oracle will also be used to determine which candidates participate in the runoff.

The winner problems for Kemeny, Young, and Dodgson are each in Θ_2^p (in fact Θ_2^p-complete (Hemaspaandra, Spakowski, and Vogel 2005; Rothe, Spakowski, and Vogel 2003; Hemaspaandra, Hemaspaandra, and Rothe 1997)), and so the complexity of each standard control action is in $\mathrm{NP}^{\Theta_2^p} = \Sigma_2^p$.[2]

Corollary 2 *For Kemeny, Young, and Dodgson elections, the complexity of each standard control action is in Σ_2^p.*

As mentioned in Brandt et al. (2015), these control problems inherit Θ_2^p-hardness from their winner problems.

We will now show that these control problems are typically Σ_2^p-complete. Our Σ_2^p-completeness results are summarized in Table 1 and we conjecture that for Kemeny, Young, and Dodgson elections, the complexity of each standard control action is Σ_2^p-complete.[3]

We mention here that there are far fewer completeness results for Σ_2^p than there are for NP (see Schaefer and Umans (2002) for a list of completeness results in the polynomial hierarchy). An important reason why proving Σ_2^p-hardness is difficult is the scarcity of "simple" Σ_2^p-complete problems to reduce from. For example, scoring versions of Kemeny, Young, and Dodgson are proven NP-hard by reductions from simple NP-complete problems such as Vertex-Cover, but prior to this paper there were no Σ_2^p-complete simple versions of Vertex-Cover that were suitable to show that related control-by-adding problems are Σ_2^p-hard.

[2]Note that $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\Theta_2^p} \subseteq \mathrm{NP}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$.

[3]De Haan (2017) shows Σ_2^p-hardness for control by adding/deleting issues for an analogue of Kemeny for judgment aggregation. Since our setting is much more restrictive, this lower bound does not at all imply our lower bound.

Table 1: Summary of our Σ_2^p-completeness results for control. Kemeny′ refers to a natural variant of Kemeny defined in (Dwork et al. 2001) and $(*)$ refers to the variant of control where the chair can delete only certain candidates.

	Adding	Deleting
Voters	Young (Thm 10) Kemeny′ (Thm 7)	Young (Thm 9)
Candidates	Kemeny (Thm 6) Dodgson (Thm 11)	Kemeny $(*)$ (Thm 5) Dodgson (Thm 11)

Below we define simple and natural Σ_2^p-complete versions of Vertex-Cover (and the analogous Independent-Set versions are also Σ_2^p-complete). We will see that these problems are particularly useful to show that our control problems are Σ_2^p-hard. Of course, we need to show that our new simple problems are Σ_2^p-hard, which is difficult. But we can then reuse our simple problems to obtain simpler Σ_2^p-hard proofs for multiple other problems.

The following problem (and its closely related Independent-Set analogue) is particularly useful to reduce to control-by-adding problems. For example, we will see that this problem quite easily reduces to Kemeny-CCAC and that the Independent-Set analogue of this problem reduces quite easily to Young-CCAV (control by adding voters).

Name: Vertex-Cover-Member-Add
Given: Graph $G = (V \cup V', E)$, set of addable vertices V', addition limit k, and vertex $\hat{v} \in V$.
Question: Does there exist a set $W \subseteq V'$ of at most k addable vertices such that $\hat{v}$ is a member of a minimum vertex cover[4] of $(V \cup W, E)$?

Theorem 3 *Vertex-Cover-Member-Add is Σ_2^p-complete.*

To show the essence of the argument of the proof of Theorem 3 and avoid some of the more finicky details of the proof, we prove that Vertex-Cover-Member-Select is Σ_2^p-complete, and then briefly discuss how this proof can be adapted for Vertex-Cover-Member-Add.[5]

Name: Vertex-Cover-Member-Select
Given: Graph $G = (V, E)$, a set $V' \subseteq V$ of deletable vertices, delete limit k, and vertex $\hat{v} \in V$.
Question: Does there exist a set $W \subseteq V'$ of at most k deletable vertices such that $\hat{v}$ is a member of a minimum vertex cover of $G - W$?

Lemma 4 *Vertex-Cover-Member-Select is Σ_2^p-complete.*

Proof. Membership in Σ_2^p is easy to see: Guess at most k deletable vertices to delete, then guess a vertex cover containing $\hat{v}$ and use the NP oracle to check that the guessed vertex cover is a minimum vertex cover.

[4]A vertex cover of a graph is a set of vertices such that every edge is incident with at least one vertex in the set.

[5]We can easily prove Vertex-Cover-Member-Select Σ_2^p-hard by reducing the Σ_2^p-complete problem Generalized-Node-Deletion (Rutenburg 1994) to it. However, this proof does not generalize to Vertex-Cover-Member-Add. For details, see the full version.

To show hardness, we reduce from the following Σ_2^p-complete problem, QSAT$_2$ (Stockmeyer 1976; Wrathall 1976): all true formulas of the form $\exists x_1 \cdots \exists x_n \neg(\exists y_1 \cdots \exists y_n \phi(x_1, \ldots, x_n, y_1, \ldots, y_n))$, where ϕ is a formula in 3cnf.[6]

We first recall the standard reduction from 3SAT to Vertex-Cover (Karp 1972). Let G be the graph constructed by this reduction on $\phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$, where ϕ is in 3cnf. Let $\phi = \psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_m$ and for each $i, 1 \leq i \leq m$, $\psi_i = c_{i,1} \vee c_{i,2} \vee c_{i,3}$. Graph G consists of $4n + 3m$ vertices: a vertex for each x_i, $\overline{x_i}$, y_i, and $\overline{y_i}$ and for each clause $i, 1 \leq i \leq m$, three vertices $c_{i,1}$, $c_{i,2}$, and $c_{i,3}$, and the following edges:

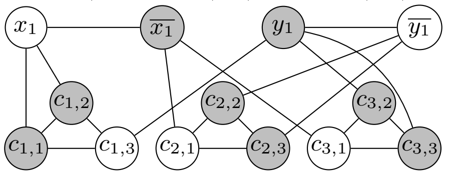
- for each $i, 1 \leq i \leq n$, the edges $\{x_i, \overline{x_i}\}$ and $\{y_i, \overline{y_i}\}$,
- for each $i, 1 \leq i \leq m$, the edges $\{c_{i,1}, c_{i,2}\}$, $\{c_{i,1}, c_{i,3}\}$, and $\{c_{i,2}, c_{i,3}\}$, (i.e., the complete graph on three vertices),
- and for each vertex $c_{i,j}$ we have an edge to its corresponding vertex candidate (e.g., if $c_{i,j} = x_t$ in ϕ then we have the edge $\{c_{i,j}, x_t\}$).

An example of this construction will follow.

Note that every vertex cover of G contains at least one of each $\{x_i, \overline{x_i}\}$, at least one of each $\{y_i, \overline{y_i}\}$, and at least two of each $\{c_{i,1}, c_{i,2}, c_{i,3}\}$, so G does not have a vertex cover of size less than $2n+2m$. The properties below follow immediately from the proof from (Karp 1972).

1. If X is a vertex cover of size $2n + 2m$, then $X \cap \{x_i, \overline{x_i}, y_i, \overline{y_i} \mid 1 \leq i \leq n\}$ corresponds to a satisfying assignment for ϕ.
2. If α is a satisfying assignment for ϕ, then there is a vertex cover X of size $2n+2m$ such that $X \cap \{x_i, \overline{x_i}, y_i, \overline{y_i} \mid 1 \leq i \leq n\}$ corresponds to this assignment.

Below we include an example of this construction given the formula $\phi = (x_1 \vee x_1 \vee y_1) \wedge (\overline{x_1} \vee \overline{y_1} \vee \overline{y_1}) \wedge (\overline{x_1} \vee y_1 \vee y_1)$.



In the figure above vertices that are in a minimum vertex cover are shaded in gray, and this corresponds to the satisfying assignment $x_1 = 0$, $y_1 = 1$ for ϕ.

For the reduction from QSAT$_2$ to Vertex-Cover-Member-Select, we construct the graph H, which is a modified version of the graph G. For each clause $i, 1 \leq i \leq m$, instead of the complete graph on three vertices, $\{c_{i,1}, c_{i,2}, c_{i,3}\}$, we add an extra vertex d_i and have the complete graph on four vertices, $\{c_{i,1}, c_{i,2}, c_{i,3}, d_i\}$, and we connect the fourth vertex d_i of each clause gadget to a special new vertex $\hat{v}$. So our graph H consists of $4n + 4m + 1$ vertices and the edges

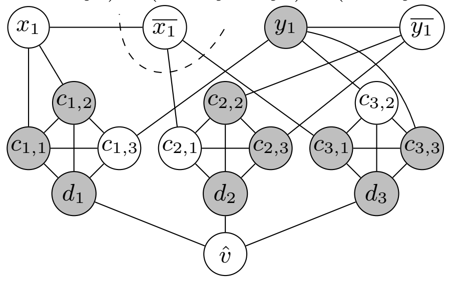
[6]Note that we have the same number of variables in each quantified block (we can simply pad to get this). Also, we pull the negation out of the formula so that the formula is in 3cnf and not in 3dnf.

as just described. Below we give the graph corresponding to the same formula as the previous example.

Note that every vertex cover of H contains at least one of each $\{x_i, \overline{x_i}\}$, at least one of each $\{y_i, \overline{y_i}\}$, and at least three of each $\{c_{i,1}, c_{i,2}, c_{i,3}, d_i\}$. So H does not have a vertex cover of size less than $2n + 3m$, and there is a vertex cover of size $2n + 3m + 1$ that includes $\hat{v}$. Note that H has the following properties.

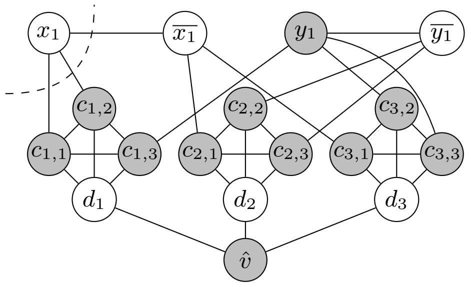
1. If X is a vertex cover of size $2n + 3m$, then $\hat{v} \notin X$ and $X \cap \{x_i, \overline{x_i}, y_i, \overline{y_i} \mid 1 \leq i \leq n\}$ corresponds to a satisfying assignment for ϕ.
2. If α is a satisfying assignment for ϕ, then there is a vertex cover of size $2n + 3m$ such that $X \cap \{x_i, \overline{x_i}, y_i, \overline{y_i} \mid 1 \leq i \leq n\}$ corresponds to this assignment.

Below we include an example of this construction given the formula $\exists x_1 \neg(\exists y_1 \phi(x_1, y_1))$, where as before $\phi = (x_1 \vee x_1 \vee y_1) \wedge (\overline{x_1} \vee \overline{y_1} \vee \overline{y_1}) \wedge (\overline{x_1} \vee y_1 \vee y_1)$.



Note that in the figure when $\overline{x_1}$ is removed (i.e., setting $x_1 = 0$) a minimum-size vertex cover (shaded in gray) is of size $n + 3m = 10$ and that $\phi(0, y_1)$ is satisfied with $y_1 = 1$.

We have repeated the same graph below, except now the vertex x_1 is removed (i.e., setting $x_1 = 1$).



The vertices shaded in gray above correspond to a minimum-size vertex cover of size $n + 3m + 1 = 11$. Note that $\phi(1, y_1)$ is not satisfiable and that this vertex cover includes $\hat{v}$.

We will show that $\exists x_1 \cdots \exists x_n \neg(\exists y_1 \cdots \exists y_n \phi(x_1, \ldots, x_n, y_1, \ldots, y_n))$ if and only if we can delete at most n vertices in $\{x_i, \overline{x_i} \mid 1 \leq i \leq n\}$ such that $\hat{v}$ is a member of a minimum vertex cover of H-after-deletion.

From the listed properties of H and the above example, it is not too hard to see that the statement above holds as long as the vertices deleted from H correspond to an assignment to the x-variables. However, it is possible for the set of deleted vertices to contain neither or both of $\{x_i, \overline{x_i}\}$. We handle these cases in the full version. ❑

To prove Theorem 3, we use a similar reduction to show that Vertex-Cover-Member-Add is Σ_2^p-hard. The main difference is that we need an edge and two vertices for each x_i and for each $\overline{x_i}$. For the proof, see the full version (Fitzsimmons et al. 2018).

Vertex-Cover-Member-Select reduces to the corresponding Kemeny control problem Kemeny-CCDC*.

Name: $\mathcal{E}$-CCDC*
Given: An election (C, V), a set of deletable candidates $D \subseteq C$, a delete limit k, and a preferred candidate $p \in C$.
Question: Does there exist a set $D' \subseteq D$ of at most k deletable candidates such that p is a winner of $(C - D', V)$ using election system $\mathcal{E}$?

Note that $\mathcal{E}$-CCDC* is more structured than $\mathcal{E}$-CCDC (where the set of deletable candidates is C), since the chair can only delete from a subset of C.

Theorem 5 *Kemeny-CCDC* is Σ_2^p-complete.*[7]

Proof Sketch. Bartholdi, Tovey, and Trick (1989) showed that Kemeny-Score is NP-hard by a reduction from Feedback-Arc-Set, which was shown to be NP-hard by Karp (1972) by a reduction from Vertex-Cover. Both these reductions are straightforward. To show the Θ_2^p-hardness of Kemeny-Winner, Hemaspaandra, Spakowski, and Vogel (2005) define Θ_2^p-complete versions of Vertex-Cover and Feedback-Arc-Set, namely Vertex-Cover-Member and Feedback-Arc-Set-Member. They show that Vertex-Cover-Member is Θ_2^p-complete. They then show that Vertex-Cover-Member reduces to Feedback-Arc-Set-Member, which then reduces to Kemeny-Winner. These two reductions are similar to the NP reductions and also straightforward. The same happens in our Σ_2^p case: Vertex-Cover-Member-Select easily and straightforwardly reduces to Feedback-Arc-Set-Member-Select, which easily and straightforwardly reduces to Kemeny-CCDC*. For details, see the full version. ❑

Vertex-Cover-Member-Add easily and similarly reduces to Feedback-Arc-Set-Member-Add (see the full version) and then to Kemeny-CCAC.

Theorem 6 *Kemeny-CCAC is Σ_2^p-complete.*

When we try to similarly show that Kemeny-CCAV is Σ_2^p-complete, it is easy to show that Feedback-Arc-Set-Member-Add-Arcs, where we add arcs instead of vertices, is Σ_2^p-complete. The problem is that in the reduction from Feedback-Arc-Set to Kemeny-Score, each arc corresponds to two voters. However, we were able to show this result for what we here call Kemeny′, the natural variant of Kemeny from (Dwork et al. 2001) where the voters do not necessarily list all of the candidates in their votes and unlisted candidates in a vote do not contribute to the distance to the Kemeny consensus and so do not increase the Kemeny score. In this case, one arc will correspond to one voter.

Theorem 7 *Kemeny′-CCAV is Σ_2^p-complete.*

We now turn to Young elections. We will explain how Independent-Set-Member-Delete, the Independent-Set analogue of Vertex-Cover-Member-Delete, which is also Σ_2^p-complete, is useful to show Young-CCDV is Σ_2^p-complete.

[7]This result holds even for four voters, using the construction from Dwork et al. (2001). For details, see the full version.

Name: Independent-Set-Member-Delete
Given: Graph $G = (V, E)$, delete limit k, and vertex $\hat{v} \in V$.
Question: Does there exist a set $W \subseteq V$ such that $\|W\| \leq k$ and $\hat{v}$ is a member of a maximum independent set of $G - W$?

Theorem 8 *Independent-Set-Member-Delete is Σ_2^p-complete.*

We reduce this problem to Young-CCDV to get the following result.

Theorem 9 *Young-CCDV is Σ_2^p-complete.*

The proofs of the above two theorems can be found in the full version. The same construction as used in this proof gives a reduction to Young-CCAV.

Theorem 10 *Young-CCAV is Σ_2^p-complete.*

Previous complexity results for Dodgson elections do not reduce from problems related to Vertex-Cover or Independent-Set. However, in Dodgson there is more flexibility in how to construct the voters in a reduction, and so we were able to directly reduce QSAT$_2$ to Dodgson-CCDC and Dodgson-CCAC, though the constructions are quite involved. The proofs can be found in the full version.

Theorem 11 *Dodgson-CCDC and CCAC are Σ_2^p-complete.*

Encoding Control Problems with ASP

When faced with a computationally difficult problem at the NP level there are several different possible ways to encode the problem to harness the power of solvers for hard problems. For example, problems in NP are often encoded for Boolean satisfiability solvers. When a problem is Σ_2^p-complete there are far fewer tools to use. However, we can encode our problem for an answer set programming (ASP) solver.

Answer set programming is a paradigm for encoding computationally difficult problems in a declarative way (see, e.g., Brewka et al. (2011)). Using modern ASP input languages like the one in the Gringo grounder (Gebser et al. 2015), which extends conventional ASP with aggregates functions like `#sum` and `#count`, we can use variable names and predicates. A descriptive naming scheme usually leads to natural encodings of problems when compared to other approaches such as encoding into Boolean satisfiability problems.

Using ASP to solve computational problems in voting was first proposed by Konczak (2006). Recent work by Charwat and Pfandler (2015) provides winner-problem encodings for many election systems, including systems with hard winner problems, and mentions encoding control problems as future work. The predicates used in their encodings are arguably self-explanatory and the use of aggregates provides succinct representations of the different voting rules they consider.

Since encoding a problem in ASP can lead to natural encodings of a problem and since ASP can encode problems in Σ_2^p (Eiter, Gottlob, and Mannila 1997), we discuss how to encode our control problems in ASP and present our complete encoding for Kemeny-CCAC. However, we mention here that although ASP encodings for NP problems (and in fact even P^{NP} problems) are fairly straightforward and common in the literature, encoding problems in Σ_2^p typically requires more advanced (and less intuitive) techniques such as saturation (Eiter, Ianni, and Krennwallner 2009). We consider how this saturation technique can be used for our problems and present an ASP encoding of Kemeny-CCAC that uses saturation. We also discuss and compare other encoding approaches for problems at this high level of complexity.

Preliminaries on Answer Set Programming

We briefly state the relevant definitions from ASP for our encoding. (See Gebser et al. (2012) for more detailed definitions.) A *disjunctive logic program* is comprised of a finite set of rules of the form $a_1 \,|\, \ldots \,|\, a_h \leftarrow b_1, b_2, \ldots, b_m, \text{not}\, b_{m+1}, \ldots, \text{not}\, b_n$ where each of $a_1, \ldots, a_h, b_1, \ldots, b_n$ are atoms and each atom is a constant or a predicate of the form $p(t_1, \ldots, t_k)$ such that $k \geq 1$ and each t_i is a constant or a variable. We indicate that p is a k-ary predicate by writing p/k. The left side of the "$\leftarrow$" is the head of the rule and the right side is the body of the rule. In the rule, "|" denotes disjunction, "," denotes conjunction, and "not" refers to default negation. Uppercase characters are used to denote variables. A *ground* program is a program that contains no variables. A subset S of the ground atoms satisfies a rule if $\{b_1, \ldots, b_m\} \subseteq S$ and $\{b_{m+1}, \ldots, b_n\} \cap S = \emptyset$ implies $a_i \in S$ for some $1 \leq i \leq h$. A *model* is a subset of the ground atoms that satisfies each rule. An *answer set* is a minimal model with respect to set inclusion.

A *fact* is a rule with no body and an *integrity constraint* is a rule with no head. So, a fact occurs in every answer set, and an integrity constraint eliminates answer sets where its body is satisfied. We additionally use choice rules with cardinality constraints, which can be used to generate subsets of ground atoms within a given bound, and aggregates such as `#count` and `#sum` that count/sum ground atoms in a statement.

It is customary to encode a decision problem as a logic program such that the answer sets of this program correspond to certificates for "yes" answers to the problem. Under this approach, a "no" answer is certified by the lack of an answer set. However, there are circumstances in which a "no" answer must be certified by an answer set (e.g., problems in coNP). The saturation technique (see, e.g., Eiter et al. (2009)) achieves this by designing a logic program that has a unique answer set including a special token atom if and only if the answer to the original decision problem is "no." This answer set also contains the set S of all atoms that would be candidate certificates for "yes" answers. Rules are added so that every time the token atom is generated, all atoms in S are generated as well, thus "saturating" the model. Informally, this allows us to encode a "coNP check" into our program, which along with an "NP guess," allows us to encode problems in Σ_2^p.

Encoding Kemeny-CCAC in ASP

We assume that the input to our problem is given as a list of facts. For $\mathcal{E}$-CCAC, our input consists of a fact for the number of registered candidates, the number of unregistered candidates, the addition limit, the preferred candidate, and facts that describe the voters. For the voters, we follow the approach used in Democratix (Charwat and Pfandler 2015), where each distinct vote $(c_1 >_i \cdots >_i c_m)$ is represented by m atoms of the form $\mathrm{p}(i, j, c)$ meaning candidate c is the jth-preferred candidate by vote i. The corresponding count is represented by $\mathrm{votecount}(i, k)$, meaning k voters have vote i.

We present our complete encoding for Kemeny-CCAC by presenting all of the rules of the encoding, split into "guess," "check," and "saturate" parts, and explaining the crucial aspects of each part. Informally, the guess part will guess a subset of unregistered candidates to add and a consensus such that the preferred candidate wins. The check part (along with the saturate part) ensures that for the guessed set of added candidates, no candidate beats the preferred candidate.

$\mathrm{preference}(1..P) \leftarrow \mathrm{prefnum}(P).$ (1)

% Registered candidates.

$\mathrm{candidate}(1..C) \leftarrow \mathrm{rcandnum}(C).$ (2)

% Unregistered candidates.

$\mathrm{ucandidate}((M+1)..(M+N)) \leftarrow \mathrm{rcandnum}(M), \mathrm{ucandnum}(N).$ (3)

% Guess a subset of at most K candidates to add.

$\{\mathrm{candidate}(C) : \mathrm{ucandidate}(C)\}K \leftarrow \mathrm{limit}(K).$ (4)

$\mathrm{candnum}(N) \leftarrow N = \#\mathtt{count}\{\mathrm{candidate}(C) : \mathrm{candidate}(C)\}.$ (5)

% Number of times candidate C is ranked below D.

$\mathrm{wrank}(P, C, D) \leftarrow \mathrm{p}(P, X, C), \mathrm{p}(P, Y, D), Y < X.$ (6)

$\mathrm{wrankC}(C, D, N) \leftarrow \mathrm{candidate}(C), \mathrm{candidate}(D),$

$\quad N = \#\mathtt{sum}\{VC, P : \mathrm{votecount}(P, VC), \mathrm{wrank}(P, C, D)\}.$ (7)

$\mathrm{position}(1..M) \leftarrow \mathrm{candnum}(M).$ (8)

% Guess a consensus. (9)

$\mathrm{gpref}(X, C) \mid \mathrm{ungpref}(X, C) \leftarrow \mathrm{position}(X), \mathrm{candidate}(C).$ (10)

$\leftarrow \mathrm{gpref}(X, C), \mathrm{gpref}(Y, C), X \neq Y.$ (11)

$\leftarrow \mathrm{gpref}(X, C), \mathrm{gpref}(X, D), D \neq C.$ (12)

$\leftarrow \mathrm{grepf}(X, C), \mathrm{ungpref}(X, C).$ (13)

% Loop checks if all possible positions for a given cand. are in ungpref.

$\mathrm{npos}(X, Y) \leftarrow \mathrm{position}(X), Y = X + 1.$ (14)

$\mathrm{countTo}(C, 1) \leftarrow \mathrm{ungpref}(1, C).$ (15)

$\mathrm{countTo}(C, X) \leftarrow \mathrm{countTo}(C, Y), \mathrm{npos}(Y, X), \mathrm{ungpref}(X, C).$ (16)

$\leftarrow \mathrm{countTo}(C, X), \mathrm{candidate}(C), \mathrm{candnum}(X).$ (17)

% In the guessed consensus $C > D$. (18)

$\mathrm{rank}(C, D) \leftarrow \mathrm{gpref}(X, C), \mathrm{gpref}(Y, D), X < Y.$ (19)

% Number of votes that disagree on C and D. (20)

$\mathrm{gwrankC}(C, D, N) \leftarrow \mathrm{rank}(C, D), \mathrm{wrankC}(C, D, N).$ (21)

$\leftarrow \mathrm{preferredCand}(X), \mathrm{gpref}(Y, X), \mathrm{position}(Y), Y \neq 1.$ (22)

Figure 1: Rules of the guess part of Kemeny-CCAC.

Figure 1 shows guess part of Kemeny-CCAC that assumes an input as described in Democratix (Charwat and Pfandler 2015), but extended with predicates for the control problem. We start by guessing (with a choice rule) a subset of at most K of the unregistered candidates to add to the election and we update the number of candidates ($\mathrm{candnum}/1$). We then define predicates $\mathrm{wrank}/3$ and $\mathrm{wrankC}/3$ to define the number of times that a candidate is ranked "worse" than another in the given election, and we guess a consensus. This generally follows what is done in Democratix (Charwat and Pfandler 2015). Specifically, we follow the naming conventions for different predicates, and rules 6, 7, 19, and 21 are from Democratix. However, our rules to guess a consensus ($\mathrm{gpref}/2$) are more involved, since we use a head disjunction.

% Guess another consensus.

$\mathrm{gpref}'(X, C) \mid \mathrm{ungpref}'(X, C) \leftarrow \mathrm{position}(X), \mathrm{candidate}(C).$ (23)

$\mathsf{sat} \leftarrow \mathrm{gpref}'(X, C), \mathrm{gpref}'(Y, C), X \neq Y.$ (24)

$\mathsf{sat} \leftarrow \mathrm{gpref}'(X, C), \mathrm{gpref}'(X, D), D \neq C.$ (25)

% Loop checks if all possible positions for a given cand. are in $\mathrm{ungpref}'$.

$\mathsf{sat} \leftarrow \mathrm{gpref}'(X, C), \mathrm{ungpref}'(X, C).$ (26)

$\mathrm{countTo}'(C, 1) \leftarrow \mathrm{ungpref}'(1, C).$ (27)

$\mathrm{countTo}'(C, X) \leftarrow \mathrm{countTo}'(C, Y), \mathrm{npos}(Y, X), \mathrm{ungpref}'(X, C).$ (28)

% Saturate if all possible positions for a given candidate are in $\mathrm{ungpref}'$,

% which means a candidate is not ranked in the guess.

$\mathsf{sat} \leftarrow \mathrm{countTo}'(C, X), \mathrm{candidate}(C), \mathrm{candnum}(X).$ (29)

% In the guessed consensus $C > D$.

$\mathrm{rank}'(C, D) \leftarrow \mathrm{gpref}'(X, C), \mathrm{gpref}'(Y, D), X < Y.$ (30)

% Number of votes that disagree on C and D. (31)

$\mathrm{gwrankC}'(C, D, N) \leftarrow \mathrm{rank}'(C, D), \mathrm{wrankC}(C, D, N).$ (32)

$\mathsf{sat} \leftarrow \#\mathtt{sum}\{M, C1, C2, pos : \mathrm{gwrankC}'(C1, C2, M);$

$\quad -N, D1, D2, neg : \mathrm{gwrankC}(D1, D2, N)\} >= 0.$ (33)

$\mathsf{sat} \leftarrow \mathrm{preferredCand}(X), \mathrm{gpref}'(1, X).$ (34)

Figure 2: Rules of the check part of Kemeny-CCAC.

Figure 2 shows the rules of the check part of Kemeny-CCAC. The check part essentially checks that given the added candidates from the guess, there is no candidate that beats the preferred candidate. Note that this is quite similar to the guess part (Figure 1). However what were integrity constraints there are now rules that generate the special saturation token sat. We describe two important aspects of the check part below.

In the check part, we start by guessing a possible consensus. This is done in the same way as the guess part of Kemeny-CCAC. In line 33 we use a $\#\mathtt{sum}$ aggregate that generates the saturation atom if a candidate other than the preferred candidate has a lower Kemeny score. Note that the use of saturation-dependent aggregates may causes undesirable behavior in the check part of an encoding and the interpretation of aggregates is solver dependent. However, the interpretation of aggregates in Clingo 4 (see Harrison et al. (2014)) allows us to use the $\#\mathtt{sum}$ aggregate in this way. We mention that our use of aggregates here follows how they are used in the saturation encodings in Abseher et al. (2015).

The final part of our encoding is the saturation step. Figure 3 shows the rules that ensure that an "incorrect" guess

$$\mathrm{gpref}'(X, C) \leftarrow \mathrm{position}(X), \mathrm{candidate}(C), \mathrm{sat}. \tag{35}$$

$$\mathrm{ungpref}'(X, C) \leftarrow \mathrm{position}(X), \mathrm{candidate}(C), \mathrm{sat}. \tag{36}$$

$$\mathrm{possibleCount}(0..X) \leftarrow \mathrm{voternum}(X). \tag{37}$$

$$\mathrm{gwrankC}'(C, D, N) \leftarrow \mathrm{candidate}(C), \mathrm{candidate}(D), \mathrm{possibleCount}(N), \mathrm{sat}. \tag{38}$$

$$\mathrm{rank}'(C, D) \leftarrow \mathrm{candidate}(C), \mathrm{candidate}(D), \mathrm{sat}. \tag{39}$$

$$\mathrm{countTo}'(C, N) \leftarrow \mathrm{candidate}(C), \mathrm{position}(N), \mathrm{sat}. \tag{40}$$

$$\leftarrow \mathrm{not\ sat}. \tag{41}$$

Figure 3: Rules of the saturation part of Kemeny-CCAC.

for the check program will cause all of the predicates that depend on the guessed consensus $\mathrm{gpref}'/2$ to have all possible values in the stable model, thus saturating the solution.

Combining the guess, check, and saturate parts we obtain our complete ASP program that is satisfiable if and only if there is a subset of unregistered candidates that can be added such that the preferred candidate is a winner.

The above encoding for Kemeny-CCAC is not as straightforward as ASP encodings for problems at the NP level, but we still have a close relationship to the definition of the problem and retain easy adaptability to minor changes in the problem description. For example, it is not difficult to change the above encoding to work for Kemeny-CCDC.

As with many declarative problem solving tools, there is a trade-off between expressibility and performance. In our tests using Clingo 4.5.4 and Preflib (Mattei and Walsh 2013) (the standard dataset of real-world preference data), we noticed the above encoding suffers from the so-called "grounding bottleneck." That is, even on instances of Kemeny-CCAC with around 10 candidates, we are faced with prohibitively large programs after all variables are instantiated. This is in part due to the use of loops in saturation (which is the standard way to replace default negation, the use of which is limited in saturation), which were found in related work on encoding Σ_2^p-complete problems in ASP to have a strong negative impact on performance (Gaggl et al. 2015). For a preliminary test of our encoding, we attempted to solve the Kemeny-CCAC problem for all elections from Preflib with 4 or more candidates having complete strict order votes, making the first candidate the preferred candidate and holding out the last fifth of the candidates as unregistered with an add limit equal to one third of the number of unregistered candidates. There were 215 elections in total, and we were able to solve 114 of them using a timeout of 1 hour and a limit of 16GB of memory. Some noteworthy instances we could solve are summarized in Table 2. Of the 101 instances that we were not able to solve, 56 ran out of memory and the rest timed out. Despite its limitations, our encoding is an important starting point for future optimization and comparison of techniques. In particular, we are interested in how techniques for overcoming the grounding bottleneck such as rewriting large rules (Bichler, Morak, and Woltran 2016) can improve the performance of our encoding.

Instance	# Reg.	# Unreg.	# Voters	Seconds	Control Possible
ED-9-2	5	2	153	1.79†	No
ED-9-1	7	2	146	368.036‡	No
ED-15-48	8	2	4	193.133‡	Yes
ED-15-78	9	3	4	78.132‡	Yes
ED-6-4	11	3	9	3.619†	No

Table 2: Details on some of the Preflib instances we were able to solve. The times reported were obtained on AMD Opteron(tm) 6180 SE (†) and AMD Opteron(tm) 6282 SE (‡) processors.

Related Encoding Approaches

We presented an encoding of Kemeny-CCAC using the saturation technique since it is the standard technique to encode Σ_2^p-complete problems in ASP and a starting point for exploring how these problems can be encoded. There are alternatives to our approach.

Eiter and Polleres (2006) address the issue of having to use advanced techniques, like saturation, when writing ASP encodings of Σ_2^p problems by providing a template "meta-interpreter" and a transformation of ASP programs that can, together, be used to integrate a program that guesses a solution to the Σ_2^p problem with a program that checks whether the guessed solution is incorrect. Their combination amounts to a "guess and check" encoding of a Σ_2^p problem.[8] However, this approach does not support ASP programs with aggregates (e.g., `#count`), which leads to more complex and somewhat less intuitive programs. To work around this issue, one could use the `lp2normal` tool (Bomanson, Gebser, and Janhunen 2014), which transforms ASP programs with aggregates into equivalent ASP programs without aggregates.

The work by Gebser et al. (2011) also addresses ASP encodings for problems with complexity higher than NP with a different metaprogramming approach that does support aggregates by using optimization rules.

The framework of *stable-unstable* semantics by Bogaerts et al. (2016) provides a similar "guess and check" strategy for solving problems in Σ_2^p. They point out that an advantage of the stable-unstable semantics is they can be easily extended to represent problems at any level of the polynomial hierarchy. In their implementation guess and check programs are each normalized, essentially discarding aggregates altogether.

It is an interesting direction for future work to determine how the performance of the above techniques and alternative paradigms for constraint satisfaction compare with saturation encodings for very hard control problems in terms of implementation and adaptability.

Future Work

In addition to the future work on ASP described at the end of the previous section, it will be interesting to see if our newly-defined simple Σ_2^p-complete problems will be useful

[8]Note this is different and much more involved than the guess and check technique for NP problems from, e.g., Eiter et al. (2009).

in proving other problems Σ_2^p-hard, in particular the remaining control cases and other election-attack problems such as manipulation for systems with hard winner problems.

Acknowledgments: We thank Andreas Pfandler for helpful conversations and we thank the referees for their helpful comments and suggestions. This work was supported in part by an NSF-GRF under grant no. DGE-1102937.

References

Abseher, M.; Bliem, B.; Charwat, G.; Dusberger, F.; and Woltran, S. 2015. Computing secure sets in graphs using answer set programming. *Journal of Logic and Computation*.

Ali, A., and Meilă, M. 2012. Experiments with Kemeny ranking: What works when? *Mathematical Social Sciences* 64(1):28–40.

Bartholdi, III, J.; Tovey, C.; and Trick, M. 1989. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare* 6(2):157–165.

Bartholdi, III, J.; Tovey, C.; and Trick, M. 1992. How hard is it to control an election? *Mathl. Comput. Modelling* 16(8/9):27–40.

Betzler, N.; Fellows, M.; Guo, J.; Niedermeier, R.; and Rosamond, F. 2008. Fixed-parameter algorithms for Kemeny scores. In *AAIM-08*, 60–71.

Betzler, N.; Bredereck, R.; and Niedermeier, R. 2014. Theoretical and empirical evaluation of data reduction for exact Kemeny Rank Aggregation. *JAAMAS* 28(5):721–748.

Bichler, M.; Morak, M.; and Woltran, S. 2016. The power of non-ground rules in answer set programming. *TPLP* 16(5-6):552–569.

Bogaerts, B.; Janhunen, T.; and Tasharrofi, S. 2016. Stable-unstable semantics: Beyond NP with normal logic programs. *TPLP* 16(5-6):570–586.

Bomanson, J.; Gebser, M.; and Janhunen, T. 2014. Improving the normalization of weight rules in answer set programs. In *JELIA-14*, 166–180.

Brandt, F.; Brill, M.; Hemaspaandra, E.; and Hemaspaandra, L. 2015. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. *JAIR* 53:439–496.

Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. 2016. *Handbook of Computational Social Choice*. Cambridge University Press.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *CACM* 54(12):92–103.

Charwat, G., and Pfandler, A. 2015. Democratix: A declarative approach to winner determination. In *ADT-15*, 253–269.

Conitzer, V.; Davenport, A.; and Kalagnanam, J. 2006. Improved bounds for computing Kemeny rankings. In *AAAI-06*, 620–626.

de Haan, R. 2017. Complexity results for manipulation, bribery and control of the Kemeny judgment aggregation procedure. In *AAMAS-17*, 1151–1159.

Dodgson, C. 1876. A method of taking votes on more than two issues. Pamphlet printed by the Clarendon Press, Oxford.

Dwork, C.; Kumar, R.; Naor, M.; and Sivakumar, D. 2001. Rank aggregation methods for the web. In *WWW-01*, 613–622.

Eiter, T., and Polleres, A. 2006. Towards automated integration of guess and check programs in answer set programming: A meta-interpreter and applications. *TPLP* 6(1-2):23–60.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems* 22(3):364–418.

Eiter, T.; Ianni, G.; and Krennwallner, T. 2009. Answer set programming: A primer. In *Reasoning Web*, 40–110.

Faliszewski, P., and Rothe, J. 2016. Control and bribery in voting. In *Handbook of Computational Social Choice*. Cambridge University Press. 146–168.

Fitzsimmons, Z.; Hemaspaandra, E.; Hoover, A.; and Narváez, D. 2018. Very hard electoral control problems. Technical Report arXiv:1811.05438 [cs.GT], arXiv.org.

Gaggl, S.; Manthey, N.; Ronca, A.; Wallner, J.; and Woltran, S. 2015. Improved answer-set programming encodings for abstract argumentation. *TPLP* 15(4-5):434–448.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.

Gebser, M.; Harrison, A.; Kaminski, R.; Lifschitz, V.; and Schaub, T. 2015. Abstract Gringo. *TPLP* 15:449–463.

Gebser, M.; Kaminski, R.; and Schaub, T. 2011. Complex optimization in answer set programming. *TPLP* 11(4-5):821–839.

Harrison, A.; Lifschitz, V.; and Yang, F. 2014. The semantics of Gringo and infinitary propositional formulas. In *KR-14*, 32–41.

Hemachandra, L. 1989. The strong exponential hierarchy collapses. *JCSS* 39(3):299–322.

Hemaspaandra, E.; Hemaspaandra, L.; and Rothe, J. 1997. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *JACM* 44(6):806–825.

Hemaspaandra, E.; Hemaspaandra, L.; and Rothe, J. 2007. Anyone but him: The complexity of precluding an alternative. *AIJ* 171(5–6):255–285.

Hemaspaandra, E.; Spakowski, H.; and Vogel, J. 2005. The complexity of Kemeny elections. *TCS* 349(3):382–391.

Karp, R. 1972. Reducibilities among combinatorial problems. In *Complexity of Computer Computations*, 85–103.

Kemeny, J. 1959. Mathematics without numbers. *Daedalus* 88:577–591.

Konczak, K. 2006. Voting theory in answer set programming. In *WLP-06*, 45–53.

Mattei, N., and Walsh, T. 2013. PrefLib: A library for preferences. In *ADT-13*, 259–270.

Meyer, A., and Stockmeyer, L. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT-72*, 125–129.

Papadimitriou, C., and Zachos, S. 1983. Two remarks on the power of counting. In *Theoretical Computer Science*, 269–276.

Rothe, J., and Schend, L. 2013. Challenges to complexity shields that are supposed to protect elections against manipulation and control: A survey. *AMAI* 68(1–3):161–193.

Rothe, J.; Spakowski, H.; and Vogel, J. 2003. Exact complexity of the winner problem for Young elections. *TOCS* 36(4):375–386.

Rutenburg, V. 1994. Propositional truth maintenance systems: Classification and complexity analysis. *AMAI* 10(3):207–231.

Schaefer, M., and Umans, C. 2002. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News* 33(3):32–49.

Stockmeyer, L. 1976. The polynomial-time hierarchy. *TCS* 3(1):1–22.

Wagner, K. 1990. Bounded query classes. *SICOMP* 19(5):833–846.

Wrathall, C. 1976. Complete sets and the polynomial-time hierarchy. *TCS* 3(1):23–33.

Young, H. 1988. Condorcet's theory of voting. *American Political Science Review* 82(2):1231–1244.