DEMOCRATIX: A Declarative Approach to Winner Determination

Günther Charwat and Andreas Pfandler

Abstract

Computing the winners of an election is an important subtask in voting and preference aggregation. The declarative nature of answer-set programming (ASP) and the performance of state-of-the-art solvers render ASP very well-suited to tackle this problem. In this work we present a novel, reduction-based approach for a variety of voting rules, ranging from tractable cases to problems harder than NP. In addition, we discuss how encodings of voting rules can be optimized and combined in our approach. The encoded voting rules are put together in the extensible tool DEMOCRATIX, which handles the computation of the winners and is also available as a web application. To learn more about the capabilities and limits of the approach, the encodings are evaluated thoroughly on real-world data as well as on random instances.

1 Introduction

Voting and preference aggregation are central topics in the field of computational social choice. Here one is interested in how opinions (or *preferences*) can be aggregated in order to obtain a collective decision. Application areas range from (political) elections to multi-agent systems, where agents have to make a joint decision over a set of alternatives. Further applications are network design and ranking algorithms for search engines (see, e.g., [11, 18]). Although voting and preference aggregation are vivid and growing research areas the number of available implementations and tools is still rather limited. In particular, there is no dedicated, freely available system that encourages experimental research in this interdisciplinary area.

In this paper we present a novel reduction-based approach for winner determination: Hereby, we express voting rules in the formalism of answer-set programming (ASP) (see, e.g., [17, 25]). ASP allows one to model problems declaratively, which not only leads to readable and maintainable code but also results in succinct encodings (compared to imperative languages). These encodings oftentimes closely resemble the mathematical definitions of the respective voting rules, thereby yielding an "executable specification". Furthermore, due to the developments of the last years, sophisticated solvers have become available for ASP ([15, 22]). All encoded voting rules are readily available in our tool DEMOCRATIX that allows the user to automatically obtain the winners of elections and also to specify further voting rules. This makes the tool especially well-suited for experimenting with new voting rules, and allows one to model new rules "hands-on" together with experts from other fields (similar to [27]). To enable a broader range of users to work with DEMOCRATIX, the tool is additionally made available as a tutorial-like web application.

So far, preference aggregation in combination with ASP has hardly ever been explored. One exception is the work of Konczak in 2006 [21], where the possible/necessary winner problem in the setting of incomplete preferences is solved for several cases which are polynomially decidable. In contrast, here we consider eleven different voting rules over fully specified preferences. For three of these rules it is harder than NP to decide whether a given candidate is among the winners. Furthermore, some work exists on implementations for specific voting rules, including Kemeny winner determination (cf. [5, 9, 10]) and approximation of Dodgson and Young elections [8]. Additionally, some commercial tools (e.g., OpenSTV [26]) are available as well as software that supports some polynomial voting rules (e.g., `http:`

//vote.sourceforge.net/). Another branch of research in the context of social choice, where reduction-based approaches have been successfully employed, is automated theorem proofing. One example is the application of the satisfiability problem (SAT) for finding strategyproof social choice functions [4] and in the area of "ranking sets of objects" [16]. Furthermore, reductions to SAT and constraint satisfaction problems (CSP) have been applied for proving, e.g., Arrow's theorem [29].

To the best of our knowledge there does not exist a uniform system that permits the declarative specification of voting rules that are harder than NP to decide. Our main contributions are the following:

- We present novel ASP encodings for a variety of voting rules, ranging from tractable (*Plurality*, *Borda* and other scoring rules, *Maximin*, *Copeland*$^\alpha$, and *Black*) to intractable (*Kemeny*, *Dodgson*, and *Young*) rules. In addition, we discuss the handling of rules with parameters, and show how encodings can be combined and optimized.

- DEMOCRATIX provides a uniform interface for all voting rules and hence can easily be extended. This makes the tool especially suitable for experimenting with further voting rules (and combinations thereof) in a declarative way. Moreover, our approach can be integrated into other software where collective decision making is required.

- The tool is made available as a web application that allows to evaluate the provided voting rules on any election with complete strict-orders given in the PrefLib format [23]. Several interactive examples help to make the tool also accessible to non-experts. Furthermore, we think the web application is useful for demonstrations and teaching, as examples and exercises can be executed and modified directly in the browser.

- We evaluate our approach using all 227 complete strict-order benchmark instances from PrefLib (see `http://www.preflib.org/` and [23]) and a collection of randomly generated elections. The benchmark results show the capabilities and limits of our approach as well as how an increase in the number of voters/candidates influences the runtime. To demonstrate how the runtime is influenced by different representations of a voting rule, we provide two alternative encodings of Kemeny's rule and compare their performance thoroughly. Results indicate that our approach works well for all tractable rules and even Kemeny's rule. For Dodgson's and Young's rule our first benchmarks give a mixed picture that depends on the structure of the instance.

The web application, the DEMOCRATIX source-code, and the encodings of the voting rules are available at:

`http://democratix.dbai.tuwien.ac.at/`

This work is structured as follows: In Section 2 we recall the required basics of voting theory and ASP; followed by Section 3, where we present our encodings. An overview of the DEMOCRATIX system and the web application is given in Section 4. After that, in Section 5, we provide an experimental evaluation of the tool. Finally, we conclude in Section 6 and provide an outlook on further developments.

2 Preliminaries

2.1 Voting Theory

Let C be a finite set of *candidates* with $|C| = m$ and $V = \{1, 2, \ldots, n\}$ a finite set of *voters*. Furthermore, let $\succ$ be a *preference relation*, i.e., a strict total order over C. The top-ranked candidate of $\succ$ is at position 1, the successor at position 2, ..., and the last-ranked

candidate is at position m. The vote of voter $i \in V$ is the preference relation $\succ_i$. A collection of preference relations $\mathcal{P} = (\succ_1, \ldots, \succ_n)$ is called a *preference profile*. A voter i *prefers* candidate c over candidate c' if $c \succ_i c'$. We denote by $\mathsf{prf}(c, c')$ the number of voters that prefer c over c'.

An *election* is given by $E = (C, V, \mathcal{P})$. A *voting rule* $\mathcal{F}$ is a mapping from an election E to a non-empty subset of the candidates $W \subseteq C$, i.e., the *winners* of the election. (We remark that, strictly speaking, "voting correspondence" would be more appropriate here. However, for sake of simplicity, we use the term "voting rule" throughout this work.) In the following we briefly recall the voting rules discussed in this paper.

Scoring rules. The class of (positional) *scoring rules* can be expressed by scoring vectors $\alpha = (\alpha_1, \ldots, \alpha_m)$, where $\alpha_i \in \mathbb{N}$ for $1 \leq i \leq m$ with $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_m$ and $\alpha_1 > \alpha_m$. To evaluate an election according to a scoring rule, the candidate ranked at position i gains α_i points. The winners of the election are the candidates having maximum score.

The well-known *plurality* rule can easily be expressed via the vector $\alpha = (1, 0, 0, \ldots, 0)$. A similar rule, the *veto* rule, can be expressed by $\alpha = (1, 1, \ldots, 1, 0)$. In another rule, *$k$-approval*, the candidates at position 1 to k gain one point each. Finally, *Borda's* rule uses the scoring vector $\alpha = (m-1, m-2, \ldots, 0)$.

Maximin. Another well-known rule is the Maximin rule (Simpson's rule). Here the Simpson score, given by $\mathit{simpson}(c) = \min_{c \neq c' \in C} \mathsf{prf}(c, c')$, has to be computed for each candidate. The winners are now exactly those candidates who have a maximum Simpson score.

Copeland. In Copeland's rule candidates are compared pairwise. In case one candidate is preferred by more voters he receives one point, the other candidate receives zero points. In case of a tie, both receive 0.5 points. The sum over the points is called the Copeland score. Winners are the candidates with maximum Copeland score. Faliszewski *et al.* [13] propose an extension of this rule, called Copeland$^\alpha$. Here, α is a rational number in $[0, 1]$. As above, if a candidate is preferred over another one in pairwise comparison he receives one point. In case of a tie, both receive α points.

Condorcet. The Condorcet winner is a candidate $c \in C$ such that for all $c' \in C \setminus \{c\}$ the condition $\mathsf{prf}(c, c') > \frac{n}{2}$ holds. Notice that there are elections without Condorcet winner.

Dodgson and Young. Since a Condorcet winner is a very favorable property there are several voting rules that try to modify an election as little as possible to obtain a Condorcet winner. There are various notions characterizing this minimality of change. One such rule is the voting rule attributed to Dodgson. For this rule, the Dodgson score is defined as the number of swaps of adjacent candidates in the votes such that there is a Condorcet winner. The winners are the Condorcet winners in elections with minimum Dodgson score. In contrast to swapping candidates, in the related Young rule votes are removed until a Condorcet winner exists.

Kemeny. Kemeny's rule is based on the distance between votes. For two votes v_1, v_2 and two candidates c_1, c_2 we define $\mathsf{disagree}(v_1, v_2, c_1, c_2)$ to be 0 if v_1 and v_2 rank the candidates c_1 and c_2 in the same way, and to be 1 otherwise. The distance between two votes v_1 and v_2 is defined as $\mathsf{dist}(v_1, v_2) = \sum_{\{c_1, c_2\} \subseteq C} \mathsf{disagree}(v_1, v_2, c_1, c_2)$. The distance between a preference relation $\succ$ and an election $E = (C, V, \mathcal{P} = (\succ_1, \ldots, \succ_n))$ is given by the Kemeny score $\mathsf{kemeny}(\succ, E) = \sum_{1 \leq i \leq n} \mathsf{dist}(\succ, \succ_i)$. A preference relation $\succ$ with minimum $\mathsf{kemeny}(\succ, E)$ is called a Kemeny consensus with respect to E. The winners according to Kemeny's rule are the top-ranked candidates in any Kemeny consensus.

Notice that determining the winner according to a scoring rule, Maximin, Copeland, as well as finding the Condorcet winner can be done in polynomial time. For the remaining rules the computational complexity is much higher: Deciding whether a candidate is a winner was shown to be Θ_2^{P}-complete for Dodgson [19], Young [28] as well as Kemeny [20]. Recall that the class Θ_2^{P} contains all problems that can be decided in polynomial time by a deterministic Turing machine using $\mathcal{O}(\log n)$ calls to an NP-oracle, where n is the input size.

2.2 Answer-Set Programming

In this section we give a brief introduction to normal logic programs under the answer-set semantics [6, 17, 25]. *Answer-set programming* (ASP) allows one to specify problems declaratively. Furthermore, powerful ASP solvers (e.g., [15, 22]) are publicly available and the ASP community is constantly working on improving the performance (witnessed, for example, by the biennial ASP competition [1]). In the following we introduce ASP, thereby restricting ourselves to the syntax and semantics relevant in this work. For a more detailed introduction see, e.g., [12, 14].

Normal programs and integrity constraints. We fix a countable set $\mathcal{U}$ of *domain elements*, also called *constants*. An *atom* is an expression $p(t_1, \ldots, t_a)$, where p is a *predicate* of arity $a \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$.

A *normal rule* r with $0 \leq k \leq n$ is of the form

$$h \leftarrow b_1, \ldots, b_k,\ not\, b_{k+1}, \ldots,\ not\, b_n.$$

The *head* of a rule r is a set $H(r) = \{h\}$, containing exactly one element. The *body* of r is $B(r) = B^+(r) \cup B^-(r)$ with $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_n\}$. Here, $h, b_1, \ldots, b_n$ are atoms, and "*not*" stands for *default negation*. An atom x is a positive literal, while $not\, x$ is a default negated literal. In the body of a rule we denote by $b(t_1; \ldots; t_l)$ the sequence of unary atoms $b(t_1), \ldots, b(t_l)$. Extending normal rules we have *integrity contraints* where $H(r) = \emptyset$ and $B(r) \neq \emptyset$.

A rule r is *safe* if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule with an empty body. A program is a finite set of safe rules. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

Answer sets. For any program π, let $\mathcal{U}_\pi$ be the set of all constants appearing in π. $Gr(\pi)$ is the set of rules r_τ obtained by applying, to each rule $r \in \pi$, all possible substitutions τ from the variables in r to elements of $\mathcal{U}_\pi$. An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program π, if each $r \in \pi$ is satisfied by I. A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\pi)$). $I \subseteq B_\mathcal{U}$ is an *answer set* of π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\pi^I = \{H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\}$.

Optimization programs. Besides normal programs, we consider the class of optimization programs, i.e., normal programs which additionally contain *weak constraints*

$$\leftsquigarrow b_1, \ldots, b_k,\ not\, b_{k+1}, \ldots,\ not\, b_n. \quad [w]$$

where all b_j are as in rules and the weight w is a positive integer variable occurring in $b_1, \ldots, b_k$ or a constant. Answer sets are minimized w.r.t. the *costs*, i.e., the sum of weights.

Aggregates and arithmetic expressions. In addition to atoms, the body of a rule can contain aggregates of the form $x \coloneqq \mathrm{aggr}_t\{p(t_1, \ldots, t_a) : p_1 : \ldots : p_l\}$ where $\mathrm{aggr} \in \{\mathrm{sum}, \mathrm{min}, \mathrm{max}\}$, $p(t_1, \ldots, t_a)$ is an atom, $p_1, \ldots, p_l$ are *conditional atoms* and t is an integer variable occurring in $t_1, \ldots, t_a$. Variable x gets assigned an integer that corresponds to the value of aggr evaluated on the values of t in all grounded instantiations of p in interpretation I, such that $p_1, \ldots, p_l$ are in I. Furthermore, $x \coloneqq \mathrm{count}\{p(t_1, \ldots, t_a)\}$ counts the number of grounded occurrences of $p(t_1, \ldots, t_a)$ in I. In addition, we allow standard relations and arithmetic expressions. All these extensions are readily supported by modern ASP solvers.

3 Winner Determination with ASP

In this section we present our approach for encoding voting rules in ASP. In particular, we focus on the following four aspects: (1) We present our novel encodings for both polynomially solvable voting rules and rules which are harder than NP. (2) We provide an alternative encoding for Kemeny's rule to demonstrate how encodings of computationally hard voting rules can be optimized. (3) We explain how voting rules with parameters (such as Copeland$^\alpha$ and k-approval) can be modeled in ASP. (4) We close this section by explaining how voting rules can be combined.

Encoding Elections. For the encodings to follow we assume the election to be given as a set of ASP facts. Let $E = (C, V, \mathcal{P})$ be an election with $C = \{c_1, \ldots, c_m\}$ and $V = \{1, \ldots, n\}$. Furthermore, let $\mathit{prefs}(\mathcal{P}) = \{\succ_1, \ldots, \succ_l\}$, $l \leq n$, denote the set of distinct preferences relations occurring in the profile $\mathcal{P}$, and $\mathit{vc}(\mathcal{P}, \succ)$ denote the number of times preference relation $\succ$ occurs in $\mathcal{P}$. For $1 \leq i \leq l$, let the preference relation $\succ_i \in \mathit{prefs}(\mathcal{P})$ be of the form $c_{i_1} \succ_i c_{i_2} \succ_i \cdots \succ_i c_{i_m}$. The input of the ASP encoding is now given as follows: Each preference relation $\succ_i$ is represented by m facts $\mathsf{p}(i, j, c_{i_j})$, where $1 \leq j \leq m$, and a single fact $\mathsf{votecount}(i, \mathit{vc}(\mathcal{P}, \succ_i))$. Additionally, three unary facts $\mathsf{voternum}(n)$, $\mathsf{candnum}(m)$, and $\mathsf{prefnum}(l)$ are added to the input. Note that it is easy to adapt this representation to handle also partial orders as well as non-anonymous voting rules.

The output of the ASP solver applied to the encoding of a voting rule together with the encoding of the election is either one or several answer sets containing winner predicates, or `UNSATISFIABLE` (in case no winner exists). For problems which are Θ_2^{P}-complete only the answer sets having minimum cost are to be considered.

3.1 ASP Encodings for Voting Rules

Scoring rules. The family of scoring rules can be expressed very naturally in ASP. Here, we start with an ASP program for Borda's rule, depicted in Encoding 1. The first rule is used to obtain the candidate relation, which contains all elements of $\{1, \ldots, m\}$. Then, we need to determine for each candidate in every preference relation the score according to his position (rule 2). Observe that this score is multiplied by VC, i.e., the number of occurrences of the preference relation in $\mathcal{P}$. Next, we sum the scores of a candidate over all votes (rule 3). Finally, in rules (4) and (5) the winner(s) are determined. Notice that in ASP "_" denotes an anonymous variable.

Encoding 1: Borda

$$\mathsf{candidate}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{1}$$

$$\mathsf{posScore}(P, C, S \cdot VC) \leftarrow \mathsf{p}(P, Pos, C), \mathsf{candnum}(M), S \coloneqq M - Pos, \mathsf{votecount}(P, VC). \tag{2}$$

$$\mathsf{score}(C, N) \leftarrow \mathsf{candidate}(C), N \coloneqq \sum_S \{\mathsf{posScore}(\_, C, S)\}. \tag{3}$$

$$\mathsf{maxScore}(M) \leftarrow M \coloneqq \max_S \{\mathsf{score}(\_, S)\}. \tag{4}$$

$$\mathsf{winner}(C) \leftarrow \mathsf{candidate}(C), \mathsf{score}(C, M), \mathsf{maxScore}(M). \tag{5}$$

It is very simple to modify Encoding 1 to specify other scoring rules, such as plurality, k-approval, and veto. Since k-approval is a voting rule with a parameter, we defer the discussion to Section 3.3. The plurality rule can be obtained by replacing rule (2) of Encoding 1 with $\mathsf{posScore}(P, C, VC) \leftarrow \mathsf{p}(P, 1, C), \mathsf{votecount}(P, VC)$. Similarly, for Veto we need to replace rule (2) with $\mathsf{posScore}(P, C, VC) \leftarrow \mathsf{p}(P, Pos, C), \mathsf{candnum}(M), Pos \neq M, \mathsf{votecount}(P, VC)$.

Maximin (Simpson's rule). This rule can nicely be expressed with help of aggregates. In Encoding 2, rules (2) and (3) are used to compute the value of function $\mathsf{prf}(c_i, c_j)$ for any

pair of distinct candidates in C, (c_i, c_j). To this end, in rule (2), we derive $\mathsf{prefer}(P, C_1, C_2)$, whenever $C_1 \succ C_2$ holds in the preference relation corresponding to the variable P. The value of function $\mathsf{prf}(c_1, c_2)$ is computed in rule (3). The Simpson score, given by the function $\mathit{simpson}(c) = \min_{c \neq c' \in C} \mathsf{prf}(c, c')$, is computed in rule (4) for each candidate. In rule (5) the maximum of $\mathit{simpson}(\cdot)$ is computed over all candidates, and the candidates having maximum Simpson score are selected in rule (6).

Encoding 2: Maximin (Simpson's rule)

$$\mathsf{candidate}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{1}$$

$$\mathsf{prefer}(P, C_1, C_2) \leftarrow \mathsf{p}(P, Pos_1, C_1), \mathsf{p}(P, Pos_2, C_2), Pos_1 < Pos_2. \tag{2}$$

$$\mathsf{preferCount}(C_1, C_2, N) \leftarrow \mathsf{candidate}(C_1; C_2), C_1 \neq C_2, \quad N \coloneqq \sum_{VC} \{\mathsf{votecount}(P, VC) : \mathsf{prefer}(P, C_1, C_2)\}. \tag{3}$$

$$\mathsf{simpson}(C, S) \leftarrow S \coloneqq \min_{N} \{\mathsf{preferCount}(C, \_, N)\}, \mathsf{candidate}(C). \tag{4}$$

$$\mathsf{maxSimpson}(M) \leftarrow M \coloneqq \max_{S} \{\mathsf{simpson}(\_, S)\}. \tag{5}$$

$$\mathsf{winner}(C) \leftarrow \mathsf{candidate}(C), \mathsf{simpson}(C, M), \mathsf{maxSimpson}(M). \tag{6}$$

Condorcet. Determining whether a given election has a Condorcet winner (and determining this winner) is a central subtask in several voting rules. The first three rules of Encoding 3 are as in the previous encoding. A candidate c cannot be a Condorcet winner if there is some other candidate c' such that $\mathsf{prf}(c, c') \leq \frac{n}{2}$. This search is encoded in rule (4) where $\mathsf{noWinner}(C)$ is derived if such a counterexample can be found for candidate C. In case no counterexample exists, we have indeed found the Condorcet winner (rule 5). The last two rules (rule 6 and 7) ensure that no answer set is returned if no Condorcet winner exists.

Encoding 3: Condorcet

$$\mathsf{candidate}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{1}$$

$$\mathsf{prefer}(P, C_1, C_2) \leftarrow \mathsf{p}(P, Pos_1, C_1), \mathsf{p}(P, Pos_2, C_2), Pos_1 < Pos_2. \tag{2}$$

$$\mathsf{preferCount}(C_1, C_2, N) \leftarrow \mathsf{candidate}(C_1; C_2), C_1 \neq C_2, \quad N \coloneqq \sum_{VC} \{\mathsf{votecount}(P, VC) : \mathsf{prefer}(P, C_1, C_2)\}. \tag{3}$$

$$\mathsf{noWinner}(C) \leftarrow \mathsf{preferCount}(C, \_, N), \mathsf{voternum}(V), N \cdot 2 \leq V. \tag{4}$$

$$\mathsf{winner}(C) \leftarrow \mathsf{candidate}(C), \mathit{not}\ \mathsf{noWinner}(C). \tag{5}$$

$$\mathsf{anyWinner} \leftarrow \mathsf{winner}(\_). \tag{6}$$

$$\leftarrow \mathit{not}\ \mathsf{anyWinner}. \tag{7}$$

Notice that only stratified default negation and no weak constraints are used in the previous encodings. Hence the encodings lie in the P fragment of ASP (data-complexity). We remark that for such programs an ASP solver can compute the unique answer set (if it exists) without backtracking. We now turn to harder voting rules, i.e., voting rules for which the problem of winner determination is Θ_2^{P}-complete. To capture these problems, the remaining encodings in this subsection make use of non-stratified default negation and weak constraints.

Kemeny. Kemeny's rule is particularly well-suited for illustrating the *guess, check & optimize* approach of ASP. Roughly speaking, we guess a preference relation and compute the Kemeny score. The Kemeny consensus is then obtained by minimizing over all guessed preference relations. The winner is the top-ranked candidate in the Kemeny consensus. In rule (1) of Encoding 4 the unary relation domain is obtained, which is used to identify candidates and positions in preferences. We then determine for each preference relation the candidates C_2 that are worse-ranked than C_1 (rule 2) and sum up the overall number of voters that do not prefer C_2 over C_1 (rule 3). Note that rules (1-3) can be computed

independently of the guess during grounding. In rules (4-9) the preference relation is guessed by assigning to each candidate exactly one position.[1] We obtain the relation rank whenever C_1 is better-ranked than C_2 in our guessed preference relation (rule 10). What remains is to compute the number of votes that disagree on C_1 being better-ranked than C_2 (rule 11). In rule (12), the sum over all N in gwrankC is computed (Kemeny score) and minimized (Kemeny consensus). A candidate ranked first in a Kemeny consensus is a winner (rule 13).

Encoding 4: Kemeny

$$\mathsf{domain}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{1}$$
$$\mathsf{wrank}(P, C_2, C_1) \leftarrow \mathsf{p}(P, Pos_1, C_1), \mathsf{p}(P, Pos_2, C_2), Pos_1 < Pos_2. \tag{2}$$
$$\mathsf{wrankC}(C_2, C_1, N) \leftarrow \mathsf{domain}(C_1; C_2), N := \sum_{VC}\{\mathsf{votecount}(P, VC) : \mathsf{wrank}(P, C_2, C_1)\}. \tag{3}$$
$$\mathsf{gpref}(Pos, C) \leftarrow \mathsf{domain}(Pos; C), \mathit{not}\ \mathsf{npref}(Pos, C). \tag{4}$$
$$\mathsf{npref}(Pos, C) \leftarrow \mathsf{domain}(Pos; C), \mathit{not}\ \mathsf{gpref}(Pos, C). \tag{5}$$
$$\leftarrow \mathsf{gpref}(Pos, C_1), \mathsf{gpref}(Pos, C_2), C_1 \neq C_2. \tag{6}$$
$$\leftarrow \mathsf{gpref}(Pos_1, C), \mathsf{gpref}(Pos_2, C), Pos_1 \neq Pos_2. \tag{7}$$
$$\mathsf{occupied}(Pos) \leftarrow \mathsf{gpref}(Pos, \_). \tag{8}$$
$$\leftarrow \mathsf{domain}(Pos), \mathit{not}\ \mathsf{occupied}(Pos). \tag{9}$$
$$\mathsf{rank}(C_1, C_2) \leftarrow \mathsf{gpref}(Pos_1, C_1), \mathsf{gpref}(Pos_2, C_2), Pos_1 < Pos_2. \tag{10}$$
$$\mathsf{gwrankC}(C_1, C_2, N) \leftarrow \mathsf{rank}(C_1, C_2), \mathsf{wrankC}(C_1, C_2, N). \tag{11}$$
$$\leftsquigarrow \mathsf{gwrankC}(\_, \_, N). \quad [N] \tag{12}$$
$$\mathsf{winner}(C) \leftarrow \mathsf{gpref}(1, C). \tag{13}$$

Dodgson. For Dodgson's rule, one could guess all $(m!)^n$ possible preference profiles, check whether there exists a Condorcet winner and minimize over the number of swaps. In order to avoid unnecessary guesses we impose the following constraints (see [3, Observation 1]). It is sufficient to allow at most one candidate shift per vote, i.e., one candidate is swapped successively i positions towards the top in the preference relation.

To allow for a simpler presentation of this encoding, we assume that the input is given in extensive form. In extensive form we do not make use of the votecount predicate to represent preferences occurring multiple times in profile $\mathcal{P}$. Instead, for a preference relation $\succ_i$ of the form $c_{i_1} \succ_i c_{i_2} \succ_i \cdots \succ_i c_{i_m}$ we introduce $vc(\mathcal{P}, \succ_i)$ many facts $\mathsf{v}(f(i,x), j, c_{i_j})$ where $1 \leq j \leq m$, $1 \leq x \leq vc(\mathcal{P}, \succ_i)$, and f is a bijection that assigns to each pair (i, x) a distinct voter in V. Notice that this conversion to extensive form can be easily realized during the preparation of the input or directly in the ASP encoding.

In Encoding 5 we first obtain the voters and the domain (positions and candidates) as in the previous encodings (rules 1-2). In rules (3-4) we guess the shifts in the votes. For a voter V the candidate at position Pos_1 will be shifted to Pos_2. At most one shift per voter (rules 5-6) to a better position (rule 7) is allowed. The preference profile is now recomputed: The candidate C_1 is moved from Pos_1 to Pos_2 (rule 8) and each candidate originally at Pos with $Pos_2 \leq Pos < Pos_1$ is shifted by one position downwards (rule 9). In the newly computed votes nv, the shifted candidates are assigned to their new positions (rule 10) and the remaining positions are filled with the respective candidates of the original vote (rules 11-12). Rules (13-18) encode the computation of the Condorcet winner, similar to Encoding 3. Finally, rule (19) minimizes over the number of swaps. Note that one shift consists of $Pos_1 - Pos_2$ elementary exchanges, i.e., swaps, of adjacent candidates.

Young's rule can be encoded quite similarly to Dodgson's rule. The basic idea is to replace rules (3-12) by a set of rules that guess the votes to be deleted. Now, in rules (13-19) we check whether this gives a Condorcet winner and minimize over the number of votes

[1] We remark that these rules can be simplified by using an additional construct, the so-called *choice rule* (see, e.g., [7]). Currently, this construct is, however, not supported by all ASP solvers.

to be deleted. In case the input is expressed in a compact way (i.e., using votecount), the encoding gets more complex. This is because we have to update votecount accordingly if a vote is removed. We have to omit a more detailed discussion due to space limitations.

Encoding 5: Dodgson

$$\mathsf{voter}(I) \leftarrow \mathsf{voternum}(N), 1 \leq I \leq N. \tag{1}$$
$$\mathsf{domain}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{2}$$
$$\mathsf{shift}(V, Pos_1, Pos_2) \leftarrow \mathsf{voter}(V), \mathsf{domain}(Pos_1; Pos_2), \mathit{not}\ \mathsf{noshift}(V, Pos_1, Pos_2). \tag{3}$$
$$\mathsf{noshift}(V, Pos_1, Pos_2) \leftarrow \mathsf{voter}(V), \mathsf{domain}(Pos_1; Pos_2), \mathit{not}\ \mathsf{shift}(V, Pos_1, Pos_2). \tag{4}$$
$$\leftarrow \mathsf{shift}(V, Pos_1, \_), \mathsf{shift}(V, Pos_1', \_), Pos_1 \neq Pos_1'. \tag{5}$$
$$\leftarrow \mathsf{shift}(V, \_, Pos_2), \mathsf{shift}(V, \_, Pos_2'), Pos_2 \neq Pos_2'. \tag{6}$$
$$\leftarrow \mathsf{shift}(V, Pos_1, Pos_2), Pos_1 \leq Pos_2. \tag{7}$$
$$\mathsf{sv}(V, Pos_2, C_1) \leftarrow \mathsf{shift}(V, Pos_1, Pos_2), \mathsf{v}(V, Pos_1, C_1). \tag{8}$$
$$\mathsf{sv}(V, \mathit{PosShift}, C) \leftarrow \mathsf{shift}(V, Pos_1, Pos_2), \mathsf{v}(V, Pos, C), Pos_2 \leq Pos, \; Pos < Pos_1, \mathit{PosShift} \coloneqq Pos + 1. \tag{9}$$
$$\mathsf{nv}(V, \mathit{PosShift}, C) \leftarrow \mathsf{sv}(V, \mathit{PosShift}, C). \tag{10}$$
$$\mathsf{occupied}(V, Pos) \leftarrow \mathsf{sv}(V, Pos, \_). \tag{11}$$
$$\mathsf{nv}(V, Pos, C_1) \leftarrow \mathsf{v}(V, Pos, C_1), \mathit{not}\ \mathsf{occupied}(V, Pos). \tag{12}$$
$$\mathsf{prefer}(V, C_1, C_2) \leftarrow \mathsf{nv}(V, Pos_1, C_1), \mathsf{nv}(V, Pos_2, C_2), Pos_1 < Pos_2. \tag{13}$$
$$\mathsf{preferCnt}(C_1, C_2, N) \leftarrow \mathsf{domain}(C_1; C_2), C_1 \neq C_2, N \coloneqq \mathrm{count}\{\mathsf{prefer}(\_, C_1, C_2)\}. \tag{14}$$
$$\mathsf{noWinner}(C) \leftarrow \mathsf{preferCnt}(C, \_, N), \mathsf{voternum}(V), N \cdot 2 \leq V. \tag{15}$$
$$\mathsf{winner}(C) \leftarrow \mathsf{domain}(C), \mathit{not}\ \mathsf{noWinner}(C). \tag{16}$$
$$\mathsf{anyWinner} \leftarrow \mathsf{winner}(\_). \tag{17}$$
$$\leftarrow \mathit{not}\ \mathsf{anyWinner}. \tag{18}$$
$$\leftsquigarrow \mathsf{shift}(\_, Pos_1, Pos_2). \quad [Pos_1 - Pos_2] \tag{19}$$

3.2 Optimizing Voting Rule Encodings

While the runtime performance for the polynomial voting rules is sufficiently good, improving the performance of the encodings of hard voting rules remains a challenging task (see Section 5 for details on the performance). In this section we exemplarily describe an alternative encoding for Kemeny's rule, which exhibits a notably better runtime behavior than Encoding 4. In Encoding 6 we apply general ASP techniques that help to reduce the runtime of the solver as well as ideas that are rather specific to Kemeny's rule.

In contrast to Encoding 4, where the preference relation is guessed directly (i.e., by setting each candidate's position explicitly), in the optimized Encoding this preference relation is obtained implicitly: The idea is to only guess the relative order for each pair of candidates within the relation, and then check whether this guess forms a valid preference relation.

In particular, within Encoding 6 we apply the following optimizations: (a) Rule (6.2) combines rules (4.2-4.3) of Encoding 4. This reduces the size of the grounding, since wrank/3 is not derived explicitly. (b) By the condition $C_1 < C_2$ in rule (6.2) only half of the candidates are compared. (c) The guess in rules (6.3-6.4) directly contains the costs (N resp. $U - N$) for a candidate C_1 being preferred over a candidate C_2. Since the weak constraint in rule (6.9) directly minimizes over these costs, the ASP solver is guided towards guessing first on prefer/3 predicates with low costs. (d) Rules (6.5-6.7) guarantee that the guess forms a valid preference relation. With xpref/2, the transitive closure over prefer/3 is obtained, and relations containing a cycle are removed. (e) Rule (6.8) is redundant but increases performance: Each candidate is either ranked before or after each other candidate.

In general, an increase in performance may be achieved by the following techniques:

(1) Remove guess-independent parts from the guess, and let them being solved by the grounder (e.g., wrankC/3). (2) It may be possible to reduce the grounding size by additional conditions (e.g., in rule 6.2). (3) For optimization problems, the solver is more performant if the costs are stated directly in the guess (e.g., rule 6.3-6.4). (4) Sometimes, redundant constraints give an increase in performance (e.g., rule 6.8). For more details on advanced modeling techniques we refer to the literature (see e.g., [14, Chapter 8]).

Encoding 6: Kemeny (Optimized)

$$\mathsf{candidate}(I) \leftarrow \mathsf{candnum}(M), 1 \leq I \leq M. \tag{1}$$

$$\mathsf{wrankC}(C_2, C_1, N) \leftarrow \mathsf{candidate}(C_1; C_2), C_1 < C_2, N \coloneqq \sum_{VC}\{\mathsf{votecount}(P, VC) : \mathsf{p}(P, Pos_1, C_1) : \mathsf{p}(P, Pos_2, C_2) : Pos_1 < Pos_2\}. \tag{2}$$

$$\mathsf{prefer}(C_2, C_1, N) \leftarrow \mathsf{wrankC}(C_2, C_1, N), \mathsf{voternum}(U), \mathit{not}\ \mathsf{prefer}(C_1, C_2, U - N). \tag{3}$$

$$\mathsf{prefer}(C_1, C_2, U - N) \leftarrow \mathsf{wrankC}(C_2, C_1, N), \mathsf{voternum}(U), \mathit{not}\ \mathsf{prefer}(C_2, C_1, N). \tag{4}$$

$$\mathsf{xpref}(C_1, C_2) \leftarrow \mathsf{prefer}(C_1, C_2, \_). \tag{5}$$

$$\mathsf{xpref}(C_1, C_3) \leftarrow \mathsf{xpref}(C_1, C_2), \mathsf{xpref}(C_2, C_3). \tag{6}$$

$$\leftarrow \mathsf{xpref}(C, C). \tag{7}$$

$$\leftarrow \mathsf{candidate}(C_1; C_2), C_1 \neq C_2, \mathit{not}\ \mathsf{xpref}(C_1, C_2), \mathit{not}\ \mathsf{xpref}(C_2, C_1). \tag{8}$$

$$\rightsquigarrow \mathsf{prefer}(\_, \_, N). \quad [N] \tag{9}$$

$$\mathsf{someBetter}(C_2) \leftarrow \mathsf{prefer}(C_1, C_2, \_). \tag{10}$$

$$\mathsf{winner}(C) \leftarrow \mathsf{candidate}(C), \mathit{not}\ \mathsf{someBetter}(C). \tag{11}$$

3.3 Voting Rules with Parameters

In some cases it is desirable to pass parameters along with the input instance to the encoding of the voting rule. These parameters are passed to the encoding in form of ASP facts.

Copeland$^\alpha$. As ASP only supports integer arithmetic, we define for Copeland$^\alpha$ two parameters, a and b. In case a candidate is preferred over another one he receives a points, in case of a tie he receives b points. The winners are the candidates with the highest sum over the points. For $a = 2$ and $b = 1$ the winners are exactly the Copeland winners.

k-approval. Another rule that is influenced by a parameter is the k-approval rule, a scoring rule where the top k candidates gain one point each. Here the parameter is given in the fact $\mathsf{kApp}(K)$. The k-approval rule can simply be implemented by replacing rule (2) in the encoding of Borda's rule (Encoding 1) by $\mathsf{posScore}(P, C, VC) \leftarrow \mathsf{p}(P, Pos, C), \mathsf{votecount}(P, VC), \mathsf{kApp}(K), Pos \leq K$.

3.4 Combining Voting Rules

Having a plethora of voting rules at hand it is a natural question to ask how one can *combine* existing voting rules. For instance, Black's rule is a combination of Condorcet and Borda's rule. Another example is the recent work of Narodytska *et al.* [24] where the properties of combinations of rules are studied.

Our approach of using ASP encodings of voting rules readily supports the combination of voting rules. Besides using a sequence of ASP solver calls, a much more elegant way is to specify a monolithic encoding. Here one has to make sure that the predicates occurring in the heads of the rules originating from different encodings are made disjoint and that the input relations do not occur in the heads. Notice that the former condition can be ensured by prefixing while the latter condition should hold in most reasonable encodings anyway.

Black. Black's rule returns the Condorcet winner if it exists, and otherwise returns the Borda winners. In Encoding 7, the winners are contained in the relations $\mathsf{winner}_{\mathrm{Cond}}(C)$ and $\mathsf{winner}_{\mathrm{Borda}}(C)$, respectively. The effort needed for "gluing" the encodings together is minimal. We add the atom computeBorda to the body of each rule that is exclusively used to compute the winners of Borda's rule. In rules (1-2) it is checked whether a Condorcet winner exists. If there is no Condorcet winner, rule (3) fires and enables the computation of the Borda winners (rule 4).

Encoding 7: Black

$$\mathsf{winner}_{\mathrm{Cond}}(C) \leftarrow \cdots \quad (1)$$
$$\mathsf{condorcet} \leftarrow \mathsf{winner}_{\mathrm{Cond}}(\_). \quad (2)$$
$$\mathsf{computeBorda} \leftarrow \mathit{not}\ \mathsf{condorcet}. \quad (3)$$
$$\mathsf{winner}_{\mathrm{Borda}}(C) \leftarrow \mathsf{computeBorda}, \ldots \quad (4)$$
$$\mathsf{winner}(C) \leftarrow \mathsf{winner}_{\mathrm{Cond}}(C). \quad (5)$$
$$\mathsf{winner}(C) \leftarrow \mathsf{winner}_{\mathrm{Borda}}(C). \quad (6)$$

Another case where combining voting rules is applicable, is the following: For voting rules that measure the distance to elections with a Condorcet winner (e.g., Dodgson, Young), it might be favorable to first check whether the instance already has a Condorcet winner. Only in case there is no Condorcet winner the grounding for the guess part has to be computed. Notice that such an encoding follows the same pattern as used for Black's rule.

4 The DEMOCRATIX System

All implemented voting rules are put together in the tool DEMOCRATIX. The application handles parsing of input instances in PrefLib format [23] to ASP facts. Internally, the ASP solver clingo (version 3.0.5) [15] is called with the input instance and the encoding of the voting rule as input. The tool is easily extendible, thereby allowing integration of new (e.g., combined) voting rules and extensions for implementations of, e.g., incomplete preferences. Furthermore, the tool is readily prepared to be used with other ASP solvers such as gringo+claspD [15] and DLV [22]. Since DEMOCRATIX is implemented in Python, it can be run both on Unix-based and Windows systems. It is licensed as open source.

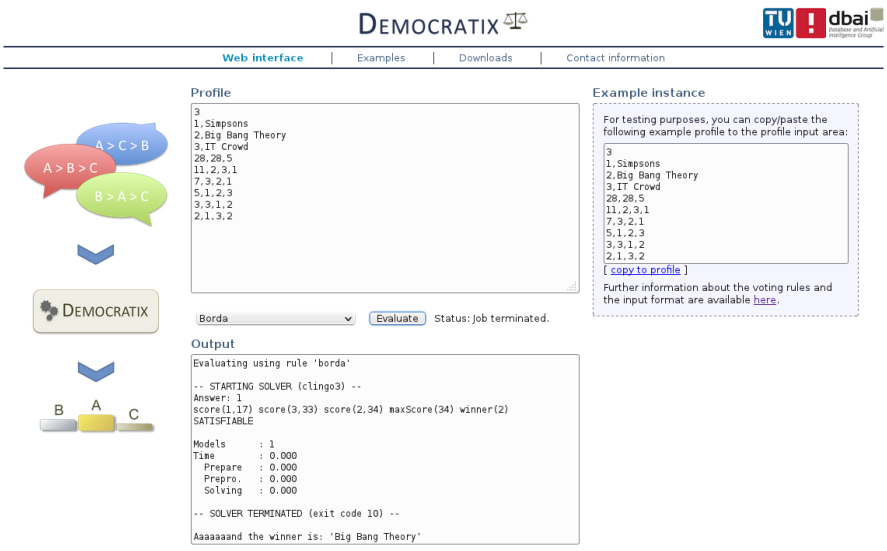

Figure 1: Screenshot of the web front-end for DEMOCRATIX.

We also provide easy access to DEMOCRATIX via a web front-end that is available at `http://democratix.dbai.tuwien.ac.at/`. A screenshot is depicted in Figure 1. There, instances from PrefLib can be evaluated directly with respect to the voting rules considered in

this work. It is also possible to submit custom instances for which the winners are computed. In the example section of the web page, the voting rules are presented and explained in a tutorial-like style. The front-end features interactive evaluation and modification of the provided examples. We believe that the web front-end is therefore particularly well-suited to make voting theory also accessible to non-experts.

Our long-term goal is to constantly extend the DEMOCRATIX system, e.g., by including further voting rules and providing support for partial-order and incomplete preference profiles. We think that our declarative ASP-based approach is the right choice to provide concise, well-readable and maintainable extensions for the system. To this end, we would also like to invite the community to contribute to the system.

5 Evaluation

Within this work we evaluate our encodings on basis of all 227 complete strict-order instances from the PrefLib library (as of February 25, 2014, available at `http://www.preflib.org/data/packs/soc.zip`) [23, 2, 26]. In particular, the PrefLib library allows us to gain a detailed insight into the runtime behavior of our tool on various kinds of instances. Furthermore, we evaluate Kemeny's rule on randomly generated instances using the *impartial culture model*. For generating those instances we used the tool "PrefLibTools-0.1". Note that our goal here is to study the capabilities and (current) limits of our general, ASP-based approach, rather than a comparison to rare rule-specific tailored implementations.

We performed benchmarks on a server with two Intel Xeon E5345 @ 2.33GHz processors and 48 GB RAM running openSUSE 11.4, kernel 2.6.37.6-24. Each run, using clingo 3.0.5, was limited to a single core and 16 GB RAM with a time limit of 10 minutes. We also tested voting rules not presented in detail in Section 3, i.e., Bucklin, Copeland and Young.

5.1 Benchmark Results for the PrefLib Data Set

Table 1 contains the results for all complete, strict-order instances of the PrefLib library.

	PrefLib (.soc)				04-163	04-182	11-002	12-001
Name	Solved	TO	MO	t(max,s)	t(s)	t(s)	t(s)	t(s)
Plurality	227	0	0	0.43	0.05	0.05	0.06	0.05
Bucklin	227	0	0	0.62	0.05	0.05	0.61	0.05
Veto	227	0	0	0.65	0.05	0.05	0.06	0.05
Borda	227	0	0	0.68	0.05	0.05	0.06	0.05
Condorcet	227	0	0	1.50	0.05	0.05	1.28	0.05
Black	227	0	0	1.52	0.05	0.05	1.37	0.06
Maximin	227	0	0	1.50	0.05	0.05	1.38	0.06
Copeland	227	0	0	1.60	0.05	0.05	1.60	0.06
Kemeny	204	22	1	TO	0.05	0.05	TO	TO
Kemeny (opt)	218	9	0	TO	0.05	0.05	TO	2.30
Dodgson	225	2	0	TO	265.44	TO	TO	67.17
Young	224	2	1	TO	TO	TO	13.85	1.14

Table 1: Results for the 227 *.soc* (strict-order, complete) instances, with the total number of solved instances, timeouts (TO) and memouts (MO). "t(max,s)" denotes the maximal (over all instances) time needed, in seconds. Detailed runtimes for instances `04-163` (m=4,n=532), `04-183` (m=4,n=578), `11-002` (m=242,n=5) and `12-001` (m=11,n=30) are given.

The results show that for all voting rules, where the problem of winner determination is in P, our ASP-based implementation is very fast. However, for problems above NP, we obtain a mixed picture: Although the problem of winner determination is Θ_2^{P}-complete for Kemeny's, Dodgson's and Young's rule, we observed different runtime behavior for our

implementations. In particular, it became evident that the optimized Kemeny encoding with 9 timeout instances is much more performant than the non-optimized encoding, where we observed 22 timeouts and 1 memout. This increase in performance is studied in detail in Section 5.2. Regarding Dodgson's and Young's rule, our encodings appear to be rather efficient on almost all PrefLib instances (with an average runtime of 1.54s (Dodgson) and 0.13s (Young) over all finished runs). However, one has to note that only 4 out of the 227 instances have no Condorcet winner. Since both encodings are designed as combined voting rules, where we first check whether there exists a Condorcet winner, for "easy" instances we can obtain the winners in a very efficient way. For a more detailed study of the runtime behavior in case there is no Condorcet winner, additional instances are necessary. If no suitable real-world instances are available, one could generate random instances parameterized by the "distance to Condorcet". This more elaborate study is, however, left for future work.

To inspect the performance on the "hard" instances, Table 1 lists the detailed runtimes for all implemented voting rules on the PrefLib instances without Condorcet winner. For Kemeny's rule, in general there are $m!$ possible preference relations to be guessed. Dodgson has $(m!)^n$ possibilities of swapping positions of candidates in the votes. Young's rule allows for 2^n combinations of removed votes from the preference profile. This difference in the number of possible combinations for obtaining the winners is reflected in the observed runtime: While the Kemeny winner(s) are determined within seconds for the instances with a low number of candidates, we were unable to solve instance `11-002` with $m = 242$ candidates. Our implementation for Dodgson's rule performed well on instance `12-001`, which comprises $m = 11$ candidates and $n = 30$ voters. Although instance `04-163` has a rather high number of votes, we obtained the Dodgson winners within our time and memory limits. Here, we were unable to determine the Young winners. We assume that this is mainly due to solver-internal heuristics as well as the fact that the grounding for Young's rule on this instance is very large. However, for instances `11-002` ($n = 5$) and `12-001` ($n = 30$) the Young implementation terminated very quickly.

5.2 Benchmark Results for the Kemeny Rule

To evaluate and compare the performance of Kemeny's rule and its alternative encoding, we generated instances (under the impartial culture model) with an increasing number of voters and candidates. Thereby it becomes clear that one can indeed benefit from the optimization techniques discussed in Section 3.2. The results are depicted in Figure 2.

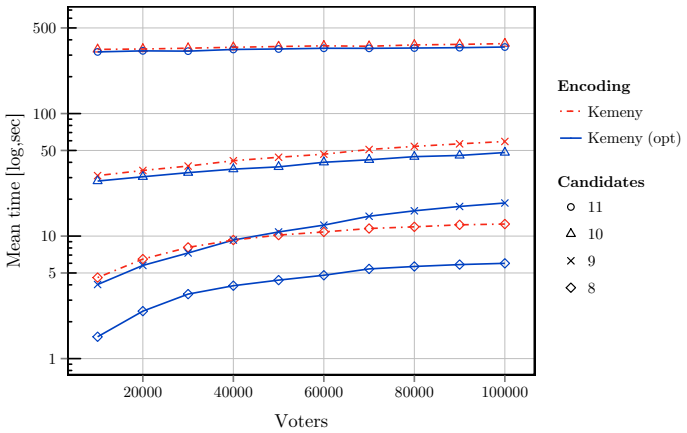

Figure 2: Benchmark results for Kemeny and its alternative (opt) variant. Each data point represents the mean runtime over 10 different randomly generated instances, in seconds, log-scale.

For both, the standard (Encoding 4) and the alternative variant (Encoding 6) it becomes evident that the runtime is exponential in the number of candidates: While solving instances with $m \leq 8$ and $n = 100000$ takes less than 12s for the standard encoding, for more candidates already approx. 60s ($m = 9$) and approx. 364s ($m = 10$) are needed. For $m = 11$, the standard variant does not terminate within our time limit. The alternative, optimized variant needs less than 6s for $m \leq 8$ and $n = 100000$, approx. 18s ($m = 9$), approx. 48s ($m = 10$) and approx. 350s ($m = 11$). Although these results show that the alternative variant is much more performant, they also indicate that a higher number of candidates drastically increases the effort for the ASP solver. On the other hand, a higher number of voters only linearly increases runtime. Overall, it becomes clear that tuning the encodings can yield a notable improvement in terms of performance.

6 Conclusion

In this work we have introduced a reduction-based approach for computing the winners of an election. To this end, we have presented ASP-encodings for a variety of well-known voting rules and explored their runtime behavior. The encodings are integrated in our tool DEMOCRATIX, that serves as a uniform and extensible system. The tool is also provided in form of an easy-to-use web application to make it accessible to a broader range of users.

The strengths of our approach clearly lie in the readability of the encodings and the extensibility of the tool. Regarding performance it turned out that problems solvable in polynomial time perform very well and even Kemeny's rule, which is Θ_2^{P}-complete, shows good runtime behavior on both real-word benchmark instances as well as on random instances. We have explained how Kemeny's rule can be optimized and observed a significant improvement in the runtime behavior. Furthermore, we have shown that even rather complicated rules like Dodgson can be modeled naturally in ASP. However, for larger instances the distance to an election having a Condorcet winner is critical for the runtime. Fortunately, any progress made in the optimization techniques for ASP will directly improve the performance of DEMOCRATIX. Conversely, our encodings of voting rules in combination with sufficiently hard preference data could serve as challenging problems for evaluating the performance of ASP solvers. Taken together, we believe that this work is a starting point for an ASP-based software tool to support experimental research in the area of voting theory and preference aggregation.

An important direction for future work is to develop encodings for other types of preferences such as incomplete preferences. Furthermore, we intend to investigate how the structure of elections (e.g., "distance to Condorcet") influences runtime and how our tool compares to rule-tailored systems. Another interesting step is the study of an ASP-based approach for problems beyond winner determination (e.g., manipulation, bribery and control) and to integrate the resulting encodings into DEMOCRATIX.

Acknowledgements

We thank the COMSOC 2014 reviewers for their constructive comments and feedback. This work was supported by the Austrian Science Fund (FWF): P25518-N23 and P25607, and by the Vienna University of Technology special fund "Innovative Projekte" (9006.09/008).

References

[1] M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, J. Pührer, C. Redl, F. Ricca, S. Schneider, M. Schwengerer, L.K. Spendier, J.P. Wallner, and G. Xiao. The fourth answer set programming competition: Preliminary report. In *Proc. LPNMR 2013*, volume 8148 of *LNCS*, pages 42–53. Springer, 2013.

[2] J. Bennett and S. Lanning. The Netflix prize. In *Proc. KDD Cup and Workshop*, 2007.

[3] N. Betzler, J. Guo, and R. Niedermeier. Parameterized computational complexity of Dodgson and Young elections. *Inf. Comput.*, 208(2):165–177, 2010.

[4] F. Brandt and C. Geist. Finding strategyproof social choice functions via SAT solving. In *Proc. AAMAS 2014*, pages 1193–1200. IFAAMAS, 2014.

[5] R. Bredereck. Fixed-parameter algorithms for computing Kemeny scores – Theory and practice. Pre-diploma thesis, Department of Mathematics and Computer Science, University of Jena, 2009.

[6] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[7] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2: 4th ASP competition official input language format. `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`, 2013.

[8] I. Caragiannis, J.A. Covey, M. Feldman, C.M. Homan, C. Kaklamanis, N. Karanikolas, A.D. Procaccia, and J.S. Rosenschein. On the approximability of Dodgson and Young elections. *Artif. Intell.*, 187:31–51, 2012.

[9] V. Conitzer, A.J. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proc. AAAI 2006*, pages 620–626. AAAI Press, 2006.

[10] A.J. Davenport and J. Kalagnanam. A computational study of the Kemeny rule for preference aggregation. In *Proc. AAAI 2004*, pages 697–702. AAAI Press, 2004.

[11] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. WWW 2001*, pages 613–622. ACM, 2001.

[12] T. Eiter, G. Ianni, and T. Krennwallner. Answer set programming: A primer. In *Proc. Reasoning Web*, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.

[13] P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *J. Artif. Intell. Res.*, 35:275–341, 2009.

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

[15] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M.T. Schneider. Potassco: The Potsdam answer set solving collection. *AI Comm.*, 24(2):107–124, 2011.

[16] C. Geist and U. Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *J. Artif. Intell. Res.*, 40:143–174, 2011.

[17] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generat. Comput.*, 9(3/4):365–386, 1991.

[18] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: The anatomy of a recommender system. In *Proc. Agents 1999*, pages 434–435. ACM, 1999.

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

[20] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theor. Comput. Sci.*, 349(3):382–391, 2005.

[21] K. Konczak. Voting theory in answer set programming. In *Proc. WLP 2006*, volume 1843-06-02 of *INFSYS Research Report*, pages 45–53. Technische Universität Wien, 2006.

[22] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[23] N. Mattei and T. Walsh. PrefLib: A library for preferences. In *Proc. ADT 2013*, volume 8176 of *LNAI*, pages 259–270. Springer, 2013.

[24] N. Narodytska, T. Walsh, and L. Xia. Combining voting rules together. In *Proc. ECAI 2012*, volume 242 of *FAIA*, pages 612–617. IOS Press, 2012.

[25] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.

[26] J. O'Neill. `www.OpenSTV.org`, 2013.

[27] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theor. Pract. Log. Prog.*, 12(03):361–381, 2012.

[28] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory Comput. Syst.*, 36(4):375–386, 2003.

[29] P. Tang and F. Lin. Computer-aided proofs of Arrow's and other impossibility theorems. *Artif. Intell.*, 173(11):1041–1053, 2009.

Günther Charwat
Institute of Information Systems
Vienna University of Technology
Vienna, Austria
Email: `gcharwat@dbai.tuwien.ac.at`

Andreas Pfandler
Institute of Information Systems
Vienna University of Technology
Vienna, Austria
Email: `pfandler@dbai.tuwien.ac.at`