HAL
archives-ouvertes.fr

Towards automatic argumentation about voting rules

Michael Kirsten, Olivier Cailloux

▶ To cite this version:

Michael Kirsten, Olivier Cailloux. Towards automatic argumentation about voting rules. 4ème conférence sur les Applications Pratiques de l'Intelligence Artificielle APIA2018, Jul 2018, Nancy, France. <hal-01830911>

HAL Id: hal-01830911
https://hal.archives-ouvertes.fr/hal-01830911

Submitted on 5 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards automatic argumentation about voting rules

Michael Kirsten[1] Olivier Cailloux[2]

[1] Dept. of Informatics, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
[2] Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, 75016 Paris, France

kirsten@kit.edu, olivier.cailloux@dauphine.fr

Abstract

Voting rules aggregate the preferences of a group to make decisions. As multiple reasonable voting rules exist, the axiomatic approach has been proposed to exhibit both their merits and paradoxical behaviors. It consists in characterizing a voting rule by a set of understandable properties called axioms. It is however a difficult task to characterize a voting rule by such axioms, and even when a proof exists, it may be difficult to understand why a specific voting rule fails to satisfy a given axiom, especially for untrained users. In this article, we present an automatic method which determines whether a given rule satisfies a set of axioms. When the rule does not satisfy an axiom, the automatic prover generates comprehensible evidence of the violation in the form of a counter-example. It can be used by non-expert users to comprehend the violation and may serve to argue in favor of other rules which satisfy the axiom. Our method is based on the software analysis technique bounded model checking*, which enables bounded verification for properties of software programs. It translates the program together with user-annotations into a reachability problem for those profiles and outcomes which adhere to our specification. The method can be applied to arbitrary voting rules; we demonstrate it on the case of the Borda axiomatization and compare the Borda rule to both the Black and the Copeland voting rules.*

Keywords

Social choice theory, bounded model checking, argumentation theory, automated reasoning.

1 Goal and motivation

Voting rules serve to select winning alternatives from a set of a priori possible alternatives, given the preferences of a set of voters. Many voting rules that appear reasonable have been proposed in the literature, however each of them shows paradoxical behavior for at least some voting situations [Arrow, 2012]. Axiomatic analysis permits to distinguish the merits of different voting rules: it consists in determining axioms, properties of voting rules that seem a priori desirable for a voting rule to have, and sets of axioms which together characterize a voting rule. A set of axioms characterizes a voting rule when there is a unique voting rule that satisfies all axioms in the set.

Our first interest in this article is, given a voting rule, to detect automatically which axioms it satisfies among a given set of axioms. Although it is not possible to do this efficiently in full generality, modern SAT- and SMT-solvers [Biere et al., 1999] enable solving this task efficiently when limiting the number of voters and alternatives. Indeed, as stated in the "small scope hypothesis" and also seen in related work on automated reasoning of voting rules, a high proportion of violations can often already be found for a comparatively small fraction of voters and alternatives [Jackson, 2006, Brandt et al., 2017]. With this prerequisite, the technique called *software bounded model checking (SBMC)* allows encoding the problem into a decidable set of logical equations, which can then be solved by a SAT- or SMT-solver. In this article we propose an automatic method that determines whether a rule satisfies an axiom within the specified bounds. We illustrate it by applying the approach to the case of Borda: we consider a set of four axioms that characterize the Borda rule, and given some rules different than the Borda rule, we find automatically which axioms from that set each rule fails to satisfy.

It is especially interesting to obtain short and readable proofs that a rule does not satisfy some axiom. This is our second objective in this article: our automatic prover outputs profiles, preferably small or satisfying some other property representing "simplicity" for a human eye, that exhibit a failure of a rule to satisfy an axiom. Such a short proof is interesting even if it is already known in the literature that the rule does not satisfy the axiom, as short proofs permit to help getting an intuitive understanding of voting rules. Furthermore, it can be used to talk about voting rules with non-expert users. Our illustration in the case of Borda permits to obtain a tool that "argues in favor" of Borda [Cailloux and Endriss, 2016] (assuming the axioms characterizing Borda of our choice are considered desirable properties of any voting rule): given any rule that is not the Borda rule, the prover finds a concrete profile which illustrates that the rule fails to comply with some of those axioms.

2 Concepts and notations

We consider an infinite set of voters $\mathscr{N}$ and a fixed finite set of alternatives $\mathscr{A}$. An election involves a subset of voters $N \subseteq \mathscr{N}$ that each express preferences over all alternatives. Those preferences are elements of $\mathscr{L}(\mathscr{A})$, the linear orders over the alternatives: connected, transitive, asymmetric binary relations over $\mathscr{A}$. Given a voter $i \in \mathscr{N}$, we write as $\succ_i \in \mathscr{L}(\mathscr{A})$ the preference of voter i. Given a set of voters N, a profile $(\succ_i)_{i\in N} = R$ is an association of each voter to her preference. The set of all possible profiles is $\bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N$.

A voting rule $f : \bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N \to \mathscr{P}^*(\mathscr{A})$, where $\mathscr{P}^*(\mathscr{A})$ represents the non-empty subsets of $\mathscr{A}$, maps each possible profile $(\succ_i)_{i\in N} \in \bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N$ to a non-empty set of winning alternatives $\emptyset \neq W \subseteq \mathscr{A}$ (sets of winners are used to account for tied winners).

An axiom is here simply defined as a property of a voting rule. One kind of axioms of interest (for their simplicity) are so-called functional properties [Beckert et al., 2016]. A functional property is one that can be represented as a relation $S \subseteq \bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N \times \mathscr{P}^*(\mathscr{A})$: given any $R \in \bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N$, $S(R) = \{A \mid (R,A) \in S\}$ contains all the admissible sets of winners given that profile. When $S(R) = \mathscr{P}^*(\mathscr{A})$, S expresses no constraint about that profile. A voting rule f satisfies the functional property S iff f picks sets of winners among the ones considered possible by S, namely, iff $f \subseteq S$ (where we view both f and S as relations). Whenever a rule f fails to satisfy a functional property, it can be proved in a very short and understandable way, by exhibiting a profile R on which f picks a wrong set of winners, $f(R) \notin S(R)$.

The property DOM is an example of a functional property.

DOM This property forbids Pareto-dominated alternatives to win. An alternative $a \in \mathscr{A}$ is Pareto-dominated in a profile $(\succ_i)_{i\in N}$ iff some alternative $a' \in \mathscr{A}$ is unanimously preferred to a in $(\succ_i)_{i\in N}$: $\forall i \in N, a' \succ_i a$. The property mandates that the rule selects winning alternatives among U_R, denoting the alternatives that are not Pareto-dominated in R: $\forall R \in \bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N$, $S_{\text{DOM}}(R) = \mathscr{P}^*(U_R)$.

Another, more general, kind of axioms we are interested in, are k-relational properties, with $k \in \mathbb{N}$ [Beckert et al., 2016]. A k-relational property $S \subseteq (\bigcup_{N\subseteq\mathscr{N}} \mathscr{L}(\mathscr{A})^N \times \mathscr{P}^*(\mathscr{A}))^k$ contains all the admissible ways of associating sets of winners to k profiles. A voting rule f satisfies a k-relational property S iff $f^k \subseteq S$, viewing again f as a relation. For example, for $k = 2$, if S contains (R_1, W_1, R_2, W_3) and (R_1, W_1, R_2, W_4), given a voting rule f such that $f(R_1) = W_1$, S declares as admissible that $f(R_2)$ be equal to W_3 or W_4.

The property REINF exemplifies a 3-relational property.

REINF Reinforcement requires that elections uniting disjoint groups of voters elect the candidates chosen by both groups, if some such candidates exist: for each $(\succ_i)_{i\in N_1}, (\succ_i)_{i\in N_2}$ with $N_1 \cap N_2 = \emptyset$ and $A \neq \emptyset$, defining A as $f((\succ_i)_{i\in N_1}) \cap f((\succ_i)_{i\in N_2})$, reinforcement imposes that $f((\succ_i)_{i\in N_1\cup N_2}) = A$.

3 Checking properties automatically

Let us start by describing the general technique of checking for software properties, before we explain how we apply it to verify whether voting rules satisfy properties. In a nutshell, we translate the proving task to a reachability problem, which the SBMC tool encodes into an instance of the SAT problem.

3.1 Checking software properties

We assume we are given an algorithm A under the form of an imperative program (for example, written in the C language), that uses some parameter values taken among a set of possible values I. An entry $i \in I$ is a list of values, one value for each such parameter: it gives a value to everything that a run of A depends on, such as its input variables, or anything that is considered non-deterministic from the point of view of A. For this reason, those parameters are qualified as "non-deterministic", to distinguish them from normal parameters used in a programming language to pass information around. (By contrast, some values can be "derived", thus, computed in A from the non-deterministic parameter values, or declared as constants in A, and both values of non-deterministic parameters or derived values can then be used as normal parameters in the program.) We are also given a software property to be checked about A, in the form $C^{\text{ant}} \Rightarrow C^{\text{cons}}$, where *ant* and *cons* stand for antecedent and consequence respectively. Both C^{ant} and C^{cons} are sets of boolean statements. A boolean statement is a statement of A that evaluates to a boolean value, for example, a statement checking that some computed intermediate value is odd. An entry i is said to satisfy a set of boolean statements iff all boolean statements in the set evaluate to true during the execution of A using the non-deterministic parameter values i, and is said to fail the set of boolean statements otherwise. The property $C^{\text{ant}} \Rightarrow C^{\text{cons}}$ requires that for all possible entries $i \in I$, *if* i satisfies C^{ant}, *then* i satisfies C^{cons}. As an example, assume A computes, given i, two intermediate integer values v_1 and v_2, and then returns a third value v_3. The property to be checked could be: *if* v_1 is negative, *then* v_2 is positive and v_3 is odd. A solver that is asked to check a software property $C^{\text{ant}} \Rightarrow C^{\text{cons}}$ thus exhaustively searches for an entry i that satisfies C^{ant} but fails C^{cons}. The property is valid *iff* it is impossible to find such an entry.

SBMC is a fully-automatic static program analysis technique used to verify whether such a software property is valid, given an algorithm and a property to be checked. It covers all possible inputs within a specified bound. It is static in the sense that programs are analyzed without executing them on concrete values. Instead, programs are symbolically executed and exhaustively checked for errors up to a certain bound, restricting the number of loop iterations to limit runs through the program to a bounded length.

This is done by unrolling the control flow graph of the program and translating it into a formula in a decidable logic that is satisfiable if and only if a program run exists which satisfies C^{ant} and fails C^{cons}. The variables in the formula are the non-deterministic parameters of A, and their possible values are taken from I.

This reduces the problem to a decidable satisfiability problem. Modern SAT or SMT-solving technology can then be used to verify whether such a program run exists, in which case an erroneous input has been found, and the run is presented to the user. If the solver cannot find such a program run, it may be either because the property is valid, or because it is invalid only for some run which exceeds the bound. In some cases, SBMC is able to infer statically which bound is sufficient to bring a definitive conclusion.

3.2 Checking voting rule properties

In our case, we are given a voting rule f implemented as a C function that returns winners given a profile, and a list of axioms. When the axiom is a functional property S, in order to check whether f satisfies S, we want to use SBMC to search for an input profile R such that $f(R) \notin S(R)$. To do this, we implement the requirements of S as a C function with annotations that indicate to the SBMC tool a set of boolean statements, constituting C^{ant}, that evaluate to true iff $f(R) \notin S(R)$. Other annotations serve to indicate non-deterministic parameters, including the input profile, and other non-deterministic parameters depending on what is necessary for our implementation to check whether a profile satisfies S. We use as C^{cons} a single boolean statement that is always false. This makes the solver search for an entry R (and possibly other parameter values) such that $f(R) \notin S(R)$.

For example, when searching for a violation of Pareto-dominance (DOM as defined in Section 2), it suffices to implement a C function F containing boolean statements that evaluate to true whenever some alternative a, a non-deterministic parameter of F, is Pareto-dominated by another alternative a', another non-deterministic parameter of F, but wins. This way, the solver will search for an entry (R, a, a') corresponding to an input profile that contains a Pareto-dominated alternative that wins, thus, an input that violates S, and will report such an input, if found.

This approach can be extended to cope for k-relational properties, where $k > 1$. We do not detail this as this is similar in spirit, only requiring to duplicate the input parameters so that the solver searches for multiple profiles as entries, and writing the function checking for S so that it cares about the relationship between multiple sets of profiles and winners.

In order to apply SBMC, we need to set bounds for the number of loop iterations. We have implemented all tested axioms and voting rules (and more) in such a way that the number of loop iterations depend only on the number of voters and alternatives. In the experiments, we set those as constants. The SBMC tool can thus automatically infer the right bounds in a syntactic pre-processing step. For more complex cases where it would not be possible to syntactically infer the bounds, e.g., when while-loops are used or the number of iterations involves some non-deterministic parameter, the bound could be specified manually.

Regarding computation time, finding an input that violates S, if such an input exists, scales much better than proving that no such input exists, as we are allowed to stop as soon as such an input is found. Moreover, our method generally scales better for smaller numbers of loop iterations, simply based on the smaller formula for which a model is searched. As we are generally looking for "small" profiles, this is most often the case. For very complex (i.e., having complex loop structures with a large number of iterations) voting rules and axioms, or for properties which are *not* violated, it may happen that the solver takes more computation time or resources than the user is willing to invest. Our method is generally well-suited for finding counter-examples to voting rule properties as most violations are exhibited for relatively small bounds. We substantiate these claims in the following section, where we exemplarily use our method to argue in favor of the Borda rule.

This approach can be generalized to search for profiles that illustrate the satisfaction of simultaneous axioms, or profiles that satisfy some axioms and not some others, etc. We have written a general program to search for such situations that cover about fifteen classical axioms[1].

For our experiments, we use CBMC 5.8 Clarke et al. [2004], an implementation of the SBMC approach for the C language, with the built-in solver based on the SAT-solver MiniSat 2.2.0 Eén and Sörensson [2003]. All experiments are performed on an Intel(R) Core(TM) i5-6500 CPU at 3.20 GHz with 4 cores and 16 GB of RAM.

3.3 A simple example

Listing 1 illustrates how we specify DOM, the "Pareto-dominance" functional property. The function is annotated with specifications in the form of assumptions (expressed by the function `__CPROVER_assume`), that represent the statements composing C^{ant}, as well as operations to model non-deterministic choice (indicated by the prefix `nondet_`) that represent what we called non-deterministic parameters in Section 3.1. These non-deterministic choice operators can be used anywhere inside the analyzed program and are also translated to the formula which is passed to the solver. The solver then searches for instantiations of these variables which lead to a violation of C^{cons}.

In the displayed function, the constants `N` and `M` denote the numbers of voters and alternatives respectively. The voters and alternatives are represented by integers from 0 to `N` -1 and from 0 to `M` -1. The function accepts as input a profile `prof`, modeled as a two-dimensional array: `prof[i][r]` is the alternative that voter `i` associates to rank `r`, where

[1] The runnable source code can be found at https://github.com/mi-ki/voting-rule-argumentation.

```
1  void dominance(uint prof[N][M], uint win[M]) {
2    uint bad = nondet_uint(), good = nondet_uint();
3    __CPROVER_assume (0 ≤ bad < M);
4    __CPROVER_assume (0 ≤ good < M);
5    __CPROVER_assume (bad ≠ good);
6    uint prefergtob[N];
7    for (uint i = 0; i < N; i++) {
8       uint rankg = M, rankb = M;
9       for (uint r = 0; r < M; r++) {
10         if (prof[i][r] == good) rankg = r;
11         if (prof[i][r] == bad) rankb = r;
12      }
13      prefergtob[i] = (rankg < rankb) ? 1 : 0;
14   }
15   for (uint i = 0; i < N; i++)
16      __CPROVER_assume (prefergtob[i]);
17   __CPROVER_assume (win[bad]);
18 }
```

Listing 1: Pareto-dominance specification for CBMC.

```
1  int main(int argc, char *argv[]) {
2     uint prof[N][M], uint win[M];
3     for (uint i = 0; i < N; i++) {
4        uint used[M];
5        for (uint a = 0; a < M; a++) used[a] = 0;
6        for (uint r = 0; r < M; r++) {
7           a = nondet_uint();
8           __CPROVER_assume (0 ≤ a < M);
9           __CPROVER_assume (!used[a]);
10          prof[i][r] = a;
11          used[a] = 1;
12       }
13    }
14    win = f(prof, N);
15    dominance(prof, win);
16    assert (0);
17    return 0;
18 }
```

Listing 2: Setup for CBMC.

`0` is the best, and `M` -1 the worst rank. The function also accepts as input a set of winning alternatives, modeled as an array `win` with one entry per alternative: `win[a]` equals `1` if the alternative `a` belongs to the set of winning alternatives, `0` otherwise. The non-deterministic parameters are `prof` (whose declaration is not shown here) and the two alternatives `bad` and `good` (Line 2). Lines 6 to 14 initialize the array `prefergtob` so that `prefergtob[i]` holds the value `1` if voter `i` prefers `good` to `bad`, `0` otherwise.

The function thus indicates to the solver that it must find two alternatives `bad` and `good`, thus integers in the suitable range (Lines 3 and 4) and different from each other (Line 5), such that everybody prefers `good` to `bad` (Lines 15 and 16), and yet the alternative `bad` is a winner of the election (Line 17). Any run which satisfies these specified statements is a valid counter-example which would prove that the profile and winners given as input violate Pareto-dominance.

We pursue the example with the setup given in Listing 2. Therein, we initialize a profile with symbolic non-deterministic values (Line 7) and restrict their ranges such that they are valid alternatives (Line 8). Furthermore, we use a helper array `used` with one entry per alternative, which is used to ensure that every ballot holds every alternative only once (Lines 5, 9 and 11). We assume a voting rule is given as a function `f` (Line 14) which gets a profile `prof` and the number of voters `N` as parameters, and returns the set of winning alternatives as output. For the experiments within this article, the implementations of `f` will follow directly from their definitions.

After calling the test methods for the properties to be checked (in this case only Pareto-dominance), we set a boolean statement that is always false (Line 16) as content of C^{cons} (recognized by the solver by the keyword `assert`), which indicate to the solver that any program run that reaches this point is a counter-example of interest to us.

4 Definitions for the experiments

We propose here to consider the axioms that characterize the Borda voting rule (Def. 1), as defined in a variant of Young's axiomatization [Cailloux and Endriss, 2016]. We also define a few axioms that are not satisfied by the Borda rule, and finally define two supplementary rules. All these concepts will be used in the experiments. (Readers knowledgeable in social choice theory can skim this section.)

Definition 1 (Borda)**.** *The Borda rule, given a profile $(\succ_i)_{i\in N}$, associates to each alternative a and voter i the score $s(a,i)$ equal to the number of alternatives that a beats in $\succ_i$, and associates to each alternative a the score $s'(a) = \sum_{i\in N} s(a,i)$. The winners are the alternatives that have the maximal score: $f_{Borda}((\succ_i)_{i\in N}) = \arg\max_{a\in\mathscr{A}} s'(a)$.*

4.1 An axiomatization of the Borda rule

We first require a few definitions.

A profile $(\succ_i)_{i\in N}$ is *elementary* iff it has exactly two voters, and if the set of alternatives can be partitioned into disjoint subsets $A^{\text{top}}, A^{\text{bottom}} \subseteq \mathscr{A}$ with $A^{\text{top}} \cup A^{\text{bottom}} = \mathscr{A}$ and $A^{\text{top}} \neq \emptyset$ such that both voters have all alternatives in A^{top} preferred to all alternatives in A^{bottom}, and the voters have inverse preferences over A^{top}, and inverse preferences over A^{bottom}. Thus, denoting the voters by 1 and 2, and denoting by $\succ|_A$ the restriction of $\succ$ to A, $\forall A \in \{A^{\text{top}}, A^{\text{bottom}}\}$: $\succ_1|_A = (\succ_2|_A)^{-1}$. Let $T(R)$ denote the "top alternatives" of an elementary profile, meaning, the set of alternatives corresponding to A^{top}. (This is legal as it is unique.)

Example 1. The profile R shown below, composed of the linear orders (a,b,c,d) and (b,a,d,c), is an elementary profile corresponding to $A^{\text{top}} = \{a,b\}$, with $\mathscr{A}$ equal to

$\{a,b,c,d\}$.

$$R = \begin{matrix} a & b \\ b & a \\ c & d \\ d & c \end{matrix}.$$

△

Given a linear order $\succ \in \mathscr{L}(\mathscr{A})$, with $\mathscr{A}$ of size m, define the cycle corresponding to $\succ$ as the set of m pairs consisting of the pairs of alternatives (a,b) such that b immediately succeeds to a in $\succ$ (the rank of b is one more than the rank of a), union the pair of alternatives (z,a) where z and a are respectively the minimal and maximal elements of $\succ$. For example, the cycle corresponding to the linear order (a,b,c) is $\{(a,b),(b,c),(c,a)\}$. Observe that, given any $\succ \in \mathscr{L}(\mathscr{A})$, the cycle corresponding to $\succ$ corresponds to exactly m linear orders in $\mathscr{L}(\mathscr{A})$. For example, the cycle $\{(a,b),(b,c),(c,a)\}$ also correspond to (b,c,a) and (c,a,b).
We say that a profile $(\succ_i)_{i\in N}$ is cyclic iff it has exactly m voters and m different linear orders, and some cycle corresponds to all its linear orders (equivalently, the linear orders in $(\succ_i)_{i\in N}$ are all those that correspond to a given cycle).

Example 2. The profile R shown below is a cyclic profile corresponding to the cycle $\{(a,b),(b,c),(c,d),(d,a)\}$ with $\mathscr{A} = \{a,b,c,d\}$.

$$R = \begin{matrix} a & b & c & d \\ b & c & d & a \\ c & d & a & b \\ d & a & b & c \end{matrix}.$$

△

Below is the axiomatization that we use for the Borda rule, composed of the axioms ELEM, CYCL, CANC and REINF. It is very similar but not identical to the axiomatization given by Young [1974]. The proof that these four axioms characterize the Borda rule is given in Cailloux and Endriss [2016].

ELEM This axiom mandates that the rule, when given any elementary profile, selects its top alternatives: $S_{\text{ELEM}}(R) = \{T(R)\}$ if R is an elementary profile and $S_{\text{ELEM}}(R) = \mathscr{P}^*(\mathscr{A})$ otherwise.

CYCL This axiom requires the rule to select all alternatives as tied winners when given any cyclic profile: $S_{\text{CYCL}}(R) = \{\mathscr{A}\}$ if R is a cyclic profile and $S_{\text{CYCL}}(R) = \mathscr{P}^*(\mathscr{A})$ otherwise.

CANC The cancellation axiom constrains the set of winners to be $\mathscr{A}$ when all pairs of alternatives (a,b) are such that a is preferred to b for as many voters as b is to a in $(\succ_i)_{i\in N}$.

REINF As defined above.

4.2 Two axioms not satisfied by Borda

Borda notoriously fails (when $|\mathscr{A}| \geq 3$) to satisfy the following two functional properties COND and MAJ, as well as the derived property WMAJ.
Given a profile $R = (\succ_i)_{i\in N}$, we say that an alternative a obtains a strict majority against a', denoted by $a\, M_R\, a'$, iff more than half of the voters prefer a to a' in $(\succ_i)_{i\in N}$: $a\, M_R\, a' \Leftrightarrow |\{i \mid a \succ_i a'\}| > |\{i \mid a' \succ_i a\}|$. An alternative is a Condorcet winner iff it obtains a strict majority against all other alternatives. Any Condorcet winner is unique.

COND This property mandates that if there is a Condorcet winner, it becomes the sole winner.

MAJ This property requires that, whenever some alternative is placed first by more than half of the voters in $(\succ_i)_{i\in N}$, it becomes the sole winner.

The Condorcet property is stronger than the majority property, i.e., $S_{\text{COND}} \subseteq S_{\text{MAJ}}$. Based on MAJ we define a weaker property, the weak majority property WMAJ, for which $S_{\text{MAJ}} \subseteq S_{\text{WMAJ}}$.

WMAJ This property requires that, whenever some alternative is placed first by more than half of the voters in $(\succ_i)_{i\in N}$, it becomes a (not necessarily unique) winner.

4.3 Two Condorcet compatible voting rules

To end this section, we define two famous voting rules that satisfy the Condorcet property.

Definition 2 (Black)**.** *The Black [1958, p. 66] rule selects the Condorcet winner if there is one, otherwise, the Borda winners.*

Given a profile $R = (\succ_i)_{i\in N}$, let $M_R(a)$ denote the set of alternatives against which a obtains a strict majority, and $M_R{}^{-1}(a)$ the set of alternatives that obtain a strict majority against a.

Definition 3 (Copeland)**.** *The Copeland [1951] rule (actually a close variant of a rule proposed by Ramon Llull in the 13th century [Colomer, 2013]), given a profile R, gives to each alternative the score $s(a) = |M_R(a)| - |M_R{}^{-1}(a)|$, and lets the alternatives with maximal score win.*

5 Experiments

In this section we first experiment with the Borda rule itself. In Section 5.1, we look at whether the solver is able to find out that the Borda rule satisfies Pareto-dominance, depending on the bounds we set on the numbers of alternatives and voters. We then check how long it takes to find an example that illustrates Pareto-dominance (one where some alternative is dominated and indeed not included among the winners). In Section 5.2, we search for counter-examples that illustrate that the Borda rule fails to satisfy the properties defined in Section 4.2.

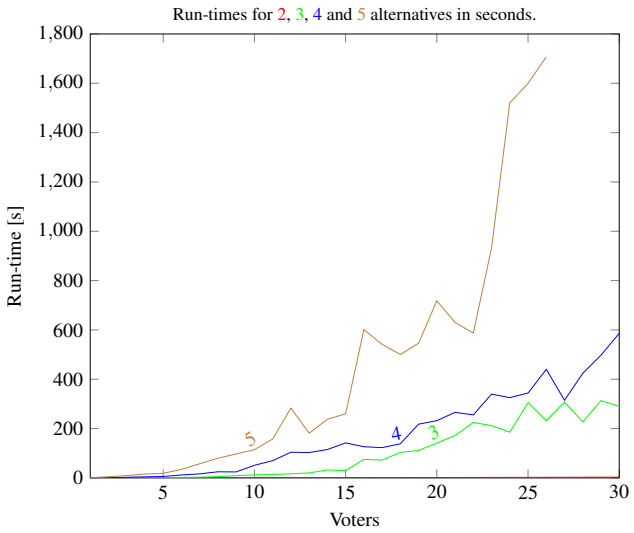

Figure 1: Run-times for the automatic verification of Pareto-dominance for the Borda voting rule.

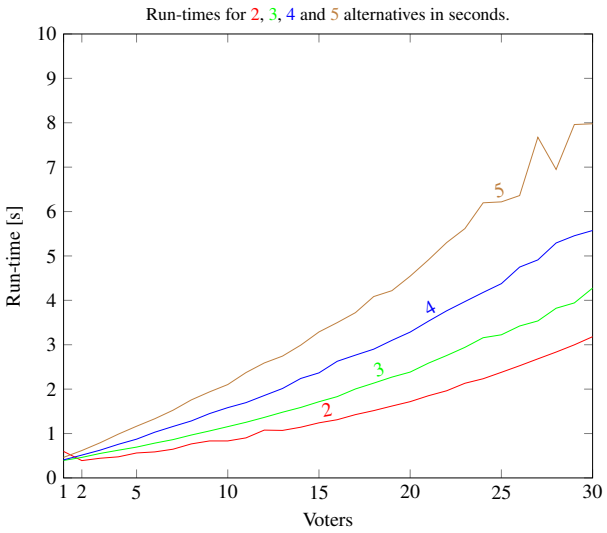

Figure 2: Run-times for finding an illustration of the dominance property for the Borda rule.

Second, we experiment with the Borda axiomatization. We suppose we are given a rule f, different from Borda (Def. 1), as a C function: in our experiments we consider the Black rule and the Copeland rule. In this simple experimental setup, we thus know that f fails to satisfy at least one of the Borda axioms from Section 4.1. For each of these rules, we illustrate that we can find out automatically which axioms f fails to satisfy, and output a short proof of this, easy for a human to inspect. We also analyze in which situations we can prove that axioms are satisfied by f, for those that f satisfies.

5.1 Borda and Pareto-dominance

In Figures 1 and 2, we illustrate the sizes of situations, i.e., numbers of voters (on the x-axis) and alternatives (each having a different plot), that we are able to analyze. In this example, as well as in the following experiments, we only consider run-times below 30 minutes to be reasonable, and stop computations which require more time than 30 minutes. Run-times are given in seconds (on the y-axis) for up to 30 voters and 5 alternatives. Figure 1 shows the run-times for the verification that Pareto-dominance (Listing 1) holds for the Borda rule (the line for 2 alternatives is almost superposed to the x-axis). Experiments for 5 alternatives and more than 26 voters took more than 30 minutes and are thus not plotted.

Figure 2 shows the run-times for finding an example situation that illustrates that Borda satisfies DOM. We used the code from Listing 1 with the only difference that we negated the statement in Line 17. The function now indicates to the solver that it must find two different alternatives `bad` and `good`, such that everybody prefers `good` to `bad`, and the alternative `bad` is not a winner of the election.

In the first experiment, the solver proves that no profiles satisfy the provided conditions. We can see that the verification is feasible for small sizes of profiles within 15 minutes. Expectedly, in situations where some profile exists that satisfies the provided conditions, the run-times are significantly smaller. They stay well below 10 seconds for this example. Both experiments indicate that our method yields reasonable run-times for at least up to five alternatives and 25 voters.

5.2 Counter-examples to Borda

We illustrate here our approach by finding counter-examples which show that the Borda rule does not satisfy the properties defined in Section 4.2 (those properties are satisfied by both the Black and the Copeland voting rule, however).

We start with the stronger Condorcet property COND. The smallest example proving that Borda fails COND can be found in less than one second for three voters and three alternatives:

$$R = \begin{matrix} c & c & b \\ b & b & a \\ a & a & c \end{matrix}.$$

For the profile R, the Borda rule elects the alternatives $\{a,c\}$ instead of the Condorcet winner c.

Whereas an isomorphic counter-example is found in less than one second for the failure of MAJ, we find two smallest examples (one regarding the number of alternatives, and another one regarding the number of voters) proving the failure of WMAJ.

For a minimal number of alternatives, we find the smallest proof for three alternatives and five voters in less than one second:

$$R = \begin{matrix} a & a & a & b & b \\ b & b & b & c & c \\ c & c & c & a & a \end{matrix}.$$

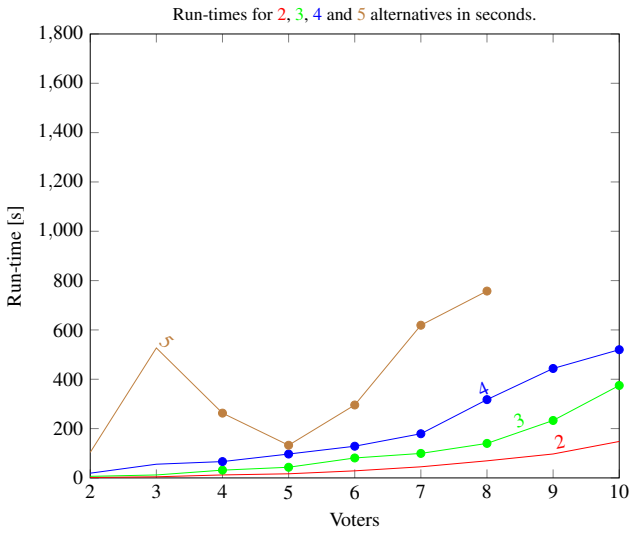

Figure 3: Run-times for the verification of REINF for the Copeland voting rule.

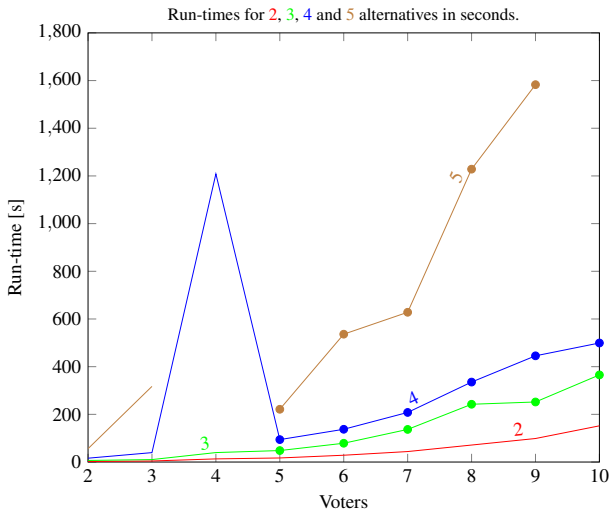

Figure 4: Run-times for the verification of REINF for the Black voting rule.

For the profile R, the Borda rule elects the alternative b instead of the majority winner a.
When searching a proof with a minimal number of voters, we find the smallest proof for four alternatives and three voters in less than one second:

$$R = \begin{matrix} d & d & c \\ c & c & a \\ a & b & b \\ b & a & d \end{matrix}.$$

For the profile R, the Borda rule elects the alternative c instead of the majority winner d. All given examples can be easily inspected manually.

5.3 Automatic comparison of Borda with other voting rules

We here want to compare the Black rule and the Copeland rule with the axiomatization of the Borda rule shown before. We first encode all four axioms in a way similar to the dominance property shown in Listing 1. The axioms ELEM, CYCL and CANC are functional properties and can thus be encoded in the very same manner. As the axiom REINF is a 3-relational property, it needs further statements relating the profiles. The three profiles are initialized with symbolic non-deterministic values as seen in Listing 2. We consider the bound N as the number of voters of the joined profile `prof`, and fix the sizes of the two sub-profiles `prof1` and `prof2` using a non-deterministic value `s` to define `prof1`'s size as `s`, and the size of `prof2` as `N - s`. We furthermore fix the profiles `prof1` and `prof2` with respect to `prof` using an array with non-deterministic values to model a mapping between the joined profile and its two sub-profiles.
Having encoded all four axioms as either functional or k-relational properties in C functions, we can now compare the Black rule and the Copeland rule with this axiomatization. We hereby rely on the "small scope hypothesis" and focus on small profiles with up to five alternatives and eight voters.

Table 1: Run-times for the verification that Black and Copeland rule each satisfy CANC.

m	n	verification time
≤ 3	≤ 8	<3 minutes
4	≤ 5 and 7	<2 minutes
4	6 and 8	>30 minutes
5	≤ 3 and 5 and 7	<1 minute
5	4 and 6 and 8	>30 minutes

Our method successfully verifies that both the Black rule and the Copeland rule satisfy the axiom ELEM in less than 17 seconds (for "small profiles" as defined above). As this axiom must only be verified for two voters, only the number of alternatives must be bounded.
For the axiom CYCL, the verification for both voting rules can be done within 10 minutes (also addressing only "small profiles" as defined above). Here, the numbers of voters are fully determined by the numbers of alternatives (CYCL only addresses profiles with an equal number of alternatives and voters).
When checking the axiom CANC, for both rules, verification is achieved within a few minutes for the cases of eight voters or less when the number of alternatives is bounded to three; five voters or less when the number of alternatives is bounded to four; and three voters or less when the number of alternatives is bounded to five. As run-times for both rules are very similar, both are depicted in Table 1. Note that CANC trivially holds for odd numbers of n, and accordingly verification within the depicted range is achieved in less than 30 seconds.

For the axiom REINF, the run-times can be seen in Figures 3 and 4 for the Copeland rule and the Black rule, respectively. Round dots indicate that violating voting situations have been found for the respective numbers of voters and alternatives. Missing parts (for the case of 5 alternatives) indicate a timeout as defined in Section 5.1. Nonetheless, the time for finding counter-examples of up to ten voters and five alternatives is significantly lower than for verifying that none exists (see, e.g., the run-times for four and five alternatives in Figure 4, comparing run-times for four and five voters). The smallest counter-examples can be found for both voting rules within less than one minute. These simultaneously serve as proofs, which are both short and easy for a human to inspect.

The smallest example (in number of alternatives) proving that the Black voting rule fails REINF has been found in 48 seconds for three alternatives and five voters:

$$R_1 = \begin{matrix} a & c \\ c & a \\ b & b \end{matrix}, \quad R_2 = \begin{matrix} c & b & a \\ b & a & c \\ a & c & b \end{matrix}.$$

The elected alternatives for the profiles R_1 and R_2 are $\{a,c\}$ and $\{a,b,c\}$ respectively. For the joined profile $R_1 \cup R_2$, the Black voting rule elects the Condorcet winner a instead of the set $\{a,c\}$ mandated by REINF.

The smallest example, both in number of alternatives and in number of voters, proving that the Copeland rule fails REINF has been found in 32 seconds for three alternatives and four voters:

$$R_1 = \begin{matrix} b & a \\ a & c \\ c & b \end{matrix}, \quad R_2 = \begin{matrix} a & b \\ b & a \\ c & c \end{matrix}.$$

The elected alternatives for the profiles R_1 and R_2 are a and $\{a,b\}$ respectively. For the joined profile $R_1 \cup R_2$, the Copeland rule elects the set $\{a,b\}$ instead of only alternative a, which would be required by the REINF property.

Both examples can be easily inspected manually.

6 Literature review

The automatic method presented within this article builds upon the techniques and concepts by Beckert et al. [2016]. It may be used to extend the argumentation framework by Cailloux and Endriss [2016].

Similarly to our approach, Brandt et al. [2017] apply automated reasoning based on SAT-solvers. They prove existing social choice theorems to obtain minimal bounds for the theorems to hold. More work on efficient applications of automated reasoning in social choice theory exists by Geist and Endriss [2011], who automatically detect many impossibility theorems. In another logic-based automatic approach, Xia [2013] incorporates axiomatic properties to a machine learning framework.

Besides fully automatic approaches, other approaches apply interactive techniques using tactical theorem provers in order to yield unbounded results. This branch was initiated by Nipkow [2009], who provides a mechanized proof for the famous Arrow's possibility theorem. Beckert et al. [2016] use optimized deductive program verification techniques to verify a number of simple voting rules with respect to classical axioms. Moreover, Dawson et al. [2015] apply interactive theorem proving to formalize and verify complex voting rules according to legal text. Another direction is pursued by Pattinson and Schürmann [2015], who encode complex voting rules in rules of higher-order logic, which serve as proof-carrying code for voting rules and may thus produce certified election results. Verification using tactical theorem provers may lead to very high confidence levels, but requires time-consuming, huge and difficult interactive proofs.

7 Conclusion and future work

In this article, we have presented and formally defined an automatic approach to argue for and against voting rules based on axiomatic social choice properties. The approach is fully automatic and based on the software analysis technique *bounded model checking*. Our case study on the voting rules Borda, Copeland and Black shows that illustrative proofs are obtained automatically in reasonable time for small numbers of voters and alternatives. This might appear surprising given the combinatorial structure of preference profiles and set-valued outcomes.

Based on the small-scope hypothesis, we argue that our approach can be used for arguing about voting rules, especially because it provides short and human-readable proofs when it detects that a voting rule fails to satisfy some axiom.

Ideas for future work include extensions to reasoning about rules that are incompletely specified. For example, we could automatically conceive an argument (in the form of an example profile and a set of winners as illustrated in the experiments) that would attack any rule that does not satisfy a given axiom. In this way, it would be possible to also argue against classes of rules, additionally to concrete rules as illustrated in this work. The approach could also be extended to automatically illustrate differences between two sets of axiomatic properties without the need for any explicit voting rule. A more ambitious extension would consist in automatically deriving proofs that a winner is a "right" winner by using several example profiles, as illustrated (using a non-automatic procedure) by Cailloux and Endriss [2016].

A longer-term objective is to mix the ideas proposed in this work and further optimization techniques [Beckert et al., 2016] with elicitation procedures in order to obtain a system that would permit to recommend a voting rule, possibly based on work in elicitation of scoring rules [Cailloux and Endriss, 2014] and on work that defines justified recommendations as those that resist counter-arguments [Cailloux and Meinard, 2017].

References

K. J. Arrow. *Social choice and individual values*. Yale University Press, New Haven, 3rd edition, 2012.

B. Beckert, T. Bormer, M. Kirsten, T. Neuber, and M. Ulbrich. Automated verification for functional and relational properties of voting rules. In *Sixth International Workshop on Computational Social Choice (COMSOC 2016)*, 2016. URL https://www.irit.fr/COMSOC-2016/proceedings/BeckertEtAlCOMSOC2016.pdf.

A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS '99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999. doi:10.1007/3-540-49059-0_14.

D. Black. *The Theory of Committees and Elections*, volume 36. Cambridge University Press, 1958. doi:10.1017/S0031819100058204.

F. Brandt, C. Geist, and D. Peters. Optimal bounds for the no-show paradox via SAT solving. *Mathematical Social Sciences*, 90:18–27, 2017. doi:10.1016/j.mathsocsci.2016.09.003.

O. Cailloux and U. Endriss. Eliciting a Suitable Voting Rule via Examples. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 183–188. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-183.

O. Cailloux and U. Endriss. Arguing about voting rules. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2016)*. IFAAMAS, 2016. URL https://dl.acm.org/citation.cfm?id=2936968.

O. Cailloux and Y. Meinard. A formal framework for deliberated judgment, 2017. URL https://arxiv.org/abs/1801.05644.

E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In K. Jensen and A. Podelski, editors, *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

J. M. Colomer. Ramon Llull: from 'Ars electionis' to social choice theory. *Social Choice and Welfare*, 40(2):317–328, 2013. doi:10.1007/s00355-011-0598-2.

A. H. Copeland. A 'reasonable' social welfare function. seminar on applications of mathematics to social sciences. In *University of Michigan Seminar on Applications of Mathematics to the Social Sciences*, 1951.

J. E. Dawson, R. Goré, and T. Meumann. Machine-checked reasoning about complex voting schemes using higher-order logic. In R. Haenni, R. E. Koenig, and D. Wikström, editors, *Proceedings of the 5th International Conference on E-Voting and Identity (VoteID 2015)*, volume 9269 of *Lecture Notes in Computer Science*, pages 142–158. Springer, 2015. doi:10.1007/978-3-319-22270-7_9.

N. Eén and N. Sörensson. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), Selected Revised Papers*, pages 502–518, 2003.

C. Geist and U. Endriss. Automated search for impossibility theorems in social choice theory: Ranking sets of objects. *Journal of Artificial Intelligence Research (JAIR)*, 40, 2011.

D. Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006. URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=10928.

T. Nipkow. Social choice theory in hol. *Journal of Automated Reasoning*, 43(3):289–304, 2009. doi:10.1007/s10817-009-9147-4.

D. Pattinson and C. Schürmann. Vote counting as mathematical proof. In B. Pfahringer and J. Renz, editors, *Proceedings of the 28th Australasian Joint Conference on Advances in Artificial Intelligence (AI 2015)*, volume 9457 of *Lecture Notes in Computer Science*, pages 464–475. Springer, 2015. doi:10.1007/978-3-319-26350-2_41.

L. Xia. Designing social choice mechanisms using machine learning. In M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems (AAMAS '13)*, pages 471–474. IFAAMAS, 2013. URL http://dl.acm.org/citation.cfm?id=2484995.

H. P. Young. An axiomatization of borda's rule. *Journal of Economic Theory*, 9(1):43 – 52, 1974. doi:10.1016/0022-0531(74)90073-8.