Ann Math Artif Intell (2013) 68:135–160
DOI 10.1007/s10472-013-9330-5

Bribery in voting with CP-nets

Nicholas Mattei · Maria Silvia Pini · Francesca Rossi · K. Brent Venable

Published online: 7 February 2013

Abstract We investigate the computational complexity of finding optimal bribery schemes in voting domains where the candidate set is the Cartesian product of a set of variables and voters use CP-nets, an expressive and compact way to represent preferences. To do this, we generalize the traditional bribery problem to take into account several issues over which agents vote, and their inter-dependencies. We consider five voting rules, three kinds of bribery actions, and five cost schemes. For most of the combinations of these parameters, we find that bribery in this setting is computationally easy.

Keywords Social choice · Voting · Bribery · CP-nets

Mathematics Subject Classifications (2010) 91B14 · 91B10

N. Mattei (✉)
NICTA and University of New South Wales, Sydney, Australia
e-mail: nicholas.mattei@nicta.com.au

M. S. Pini
Department of Information Engineering, University of Padova, Padova, Italy
e-mail: pini@dei.unipd.it

F. Rossi
Department of Mathematics, University of Padova, Padova, Italy
e-mail: frossi@math.unipd.it

K. B. Venable
Department of Computer Science, Tulane University and IHMC, New Orleans LA, USA
e-mail: kvenabl@tulane.edu

1 Introduction

It is often natural to express group decision problems as the combination of a sequence of decisions. This method is used in many settings, from the United States Congress (specifically, votes for amendments to a bill) to a group of friends deciding what appetizer, main course, desert, and wine should be served for a group meal [9, 27]. In all these cases, agents express preferences and vote on parts of the overall decision to be taken. Moreover, agents may have dependent preferences: the choice of wine may depend on the choice of main course for the meal. We consider a scenario where agents use the CP-net formalism, which allows them to compactly represent their possibly conditional preferences over many issues [7].

Bribery and manipulation directly question the security and robustness of an election system. They are ways that agents (outside or inside an election) can use to affect the election's outcome. In particular, the bribery problem regards scenarios in which an outside agent with a limited budget attempts to affect the outcome of an election by paying some of the agents to change their preferences so that a particular candidate wins. This problem was introduced to computational social choice in [20, 21] and has become a metric by which the security of election rules is judged [15, 28]: if it is computationally difficult to find an optimal bribery in elections decided by a certain voting rule, we can say that the rule is resistant to bribery. The literature, however, is sparse when elections have combinatorially structured domains (see for example [38] for a study of strategy-proof sequential voting rules).

When voting is structured as the combination of several decisions, there are two natural methods to employ when selecting a winner. We investigate multiple individual rules within two winner determination strategies.

- Sequential Methods: A sequential method aggregates each agent's preference on an issue-by-issue basis. This is akin to how amendments to a bill are passed in the US Congress [9]. The final decision is the sum of all the sequential decisions. In this paper we consider the sequential majority (SM) rule.
- One-step Methods: A one-step method aggregates the agents' votes over the set of all combinations of value assignments to all issues. This would be the method where a group of friends votes on a complete meal for some shared group meal [27]. With a one-step method we can incorporate several individual voting rules. In this section we consider one-step plurality (OP), one-step veto (OV), and one-step k-approval (OK).

Unlike previous studies on bribery, when we use CP-nets to encode the preferences of a voter, the encoding does not necessarily lend itself to the convention that the cost of bribery is either a fixed constant for all voters, a fixed constant on a per-voter basis, or related to the number of swaps in the candidate ordering [18, 21]. In fact, we do not always want to work on the outcome ordering, since it is exponentially large, but on the CP-net. Since a single flip in a CP-net preference statement (cp-statement) can lead to a large change in the outcome ordering, we consider five different bribery cost schemes:

- C_{EQUAL}: Any amount of change in a CP-net is a unit cost.
- C_{FLIP}: The cost of bribery is the number of variable flips in the CP-net.
- C_{LEVEL}: The cost is the number of variable flips weighted by variable position within the CP-net.

- C_{ANY}: The cost is the number of variable flips weighted by a specific cost per variable.
- C_{DIST}: The cost is based on the distance (number of worsening flips) between the current optimal outcome and the new outcome to be made optimal.

We show that SM bribery is easy except when we use C_{EQUAL}. However, when voters are weighted, bribery for SM is always difficult. For OP and OV, bribery is always easy, except for two cases, when C_{ANY} or C_{LEVEL} are combined with the possibility of making flips in preference statements of dependent variables. In these cases, we don't have a formal result although we believe the problem is difficult. Bribery is easy also for OK when k is a power of 2.

The paper is organized as follows. In Section 2 we provide the basic notions about voting theory and bribery, as well as a definition of CP-nets and their uses for compactly expressing preferences in combinatorial domains. In Section 3 we precisely define our domain and formulate the general bribery problem within our model. In the sections following Section 3, we give our results about the computational complexity of answering the bribery problem in our models. These sections, which are broken up according to voting rules and aggregation methods, provide results for voting rules such as sequential majority, plurality, k-approval, and veto. Section 8 summarizes the results of the paper and gives some hints for future work.

A very preliminary and informal version of some parts of this paper appeared in [29].

2 Basic notions

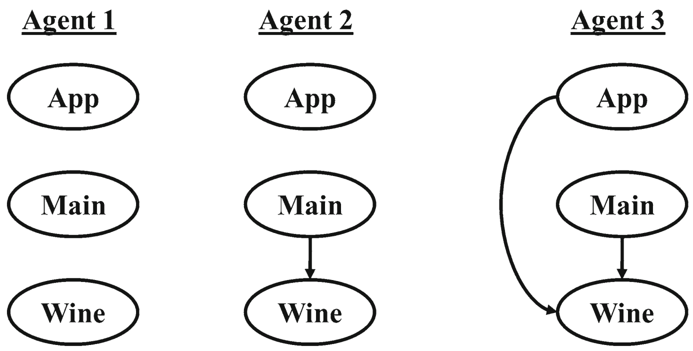
2.1 CP-nets

A major challenge for computational social choice is describing agent preferences in domains where there is a large number of candidates. Artificial Intelligence provides several formalisms to do this, including CP-nets [7], the one we consider in this work. CP-nets are a graphical model for compactly representing conditional and qualitative preference relations. CP-nets are sets of *ceteris paribus* preference statements (cp-statements). For instance, the cp-statement "I prefer red wine to white wine if meat is served". asserts that, given two meals that differ only in the kind of wine served and both containing meat, the meal with red wine is preferable to the meal with white wine.

Formally, a CP-net has a set of issues or variables $M = \{x_1, \ldots, x_m\}$, and each issue has a finite domain $D(x_1), \ldots, D(x_m)$. Each issue x has a set of parent issues $Pa(x)$ whose assignment can affect the ordering over the values of x. This set of dependencies defines a graph where each x has $Pa(x)$ as its immediate predecessors. In general, CP-nets allow for cyclic dependencies within the implied graph. However, in this paper we only allow acyclic dependency graphs. Issue x is an independent issue if its set of parent issues is empty; otherwise it is a dependent issue. Given the implied graph structure an agent specifies a strict total order over the values of each x for each complete assignment of the variables in $Pa(x)$. This is done by using so-called cp-statements of the form $x_1 = v_1, \ldots, x_m = v_m : x = a_1 > \cdots > x = a_k$, where $Pa(x) = x_1, \ldots, x_m$, $D(x) = a_1, \ldots, a_k$ and $>$ is a total order over $D(x)$. A CP-net is *compact* if the maximum number of parents of a feature is bounded from above by a constant. This implies that the number of outcomes (that is, assignments of values to variables) is exponential in the representation size of the CP-net.

Example 1 Figure 1 shows three CP-nets, belonging to three different agents (1, 2, and 3). They all have the same set of three variables (also called issues) $\{App, Main, Wine\}$. The domains for the variables are $D(App) = \{Soup, Salad\}$, $D(Main) = \{Fish, Meat\}$, and $D(Wine) = \{Red, White\}$. Agent 1 has no preferential dependencies and therefore none of his variables have parent variables. The preferences of agent 2 on wine are dependent on his assignment for the main course. The encoding of such preferences model the statement "I prefer red wine to white if meat is served". So, for agent 2, given two meals that differ only in the kind of wine, the meal with red wine is more preferred. Agent 3 has a more complex set of dependencies in his CP-net. Recall that, for each variable, a complete assignment must be provided for all assignments of the parent variables. This is why the preferences of agent 3 are so much more complex. Since the *Wine* issue has two parents, agent 3 must specify a wine preference for all possible combinations of App and $Main$.

We consider issues with only binary domains in this paper. For a given issue A, we denote A's binary domain as a and $\overline{a}$ with an individual cp-statement of $a > \overline{a}$. Given a set of issues $\{A, B\}$ where the assignment of B is dependent on the assignment of A, we write our cp-statements as: $a > \overline{a}$, $a : \overline{b} > b$, and $\overline{a} : b > \overline{b}$. These statements imply the dependency graph and explicitly state that $A = a$ is unconditionally preferred to $A = \overline{a}$ and that $B = \overline{b}$ is preferred when $A = a$. A complete assignment to all issues is called an *outcome* or a *candidate*. When examining binary domains, CP-nets allow us to compactly express preferences over an outcome space of cardinality exactly 2^m, if and only if m is the number of issues.



App	Soup > Salad
Main	Meat > Fish
Wine	Red > White

App	Salad > Soup
Main	Fish > Meat
Wine	Fish: White > Red Meat: Red > White

App	Salad > Soup
Main	Fish > Meat
Wine	Salad, Fish: White > Red Salad, Meat: Red > White Soup, Meat: Red > White Soup, Fish: Red > White

Fig. 1 Three CP-nets expressing an O-legal profile over three binary variables

In general, finding the optimal (most preferred) outcome of a CP-net is NP-hard [7]. However, in acyclic CP-nets, there is only one optimal outcome and this outcome can be found in linear time by sweeping through the CP-net in topological order and assigning the most preferred values according to the cp-statements. Consider Agent 3 in the example illustrated in Fig. 1. Agent 3 would choose $App = Salad$ and $Main = Fish$. After these two are set we investigate the cp-statement for the last variable and select the appropriate value $Wine = White$ to find his top outcome.

Given an outcome for an agent, a *worsening flip* is a change in the value of an issue to a less preferred value according to the relevant cp-statement for that issue. In general, one outcome α is better than another outcome β ($\alpha \succ \beta$) if and only if there is a chain of worsening flips from α to β. This definition induces a preorder (reflexive and transitive relation) over the outcomes, which is a partial order (reflexive, transitive, and anti-symmetric relation) if the dependency graph of the CP-net is acyclic.

Example 2 Figure 2 illustrates the partial order over all the outcomes defined by the CP-net of agent 1. The most preferred outcome according to this CP-net is $\{Soup, Meat, Red\}$. The arrows from this outcome represent all the possible worsening flips. Traversing any of the edges from one outcome to another down the induced graph of assignments means we are moving from a better to a worse outcome. Note that, since there are no dependencies and we have not defined any relationship between the independent variables, the nodes of the graph at each level are incomparable.

In an acyclic CP-net, it is also computationally easy, given an outcome, to find the next best outcome in a linearization of the induced outcome ordering [8]. In a CP-net some outcomes may not be ordered, since a CP-net defines a partial order over the outcomes, so some assignments may be incomparable. When a strict total

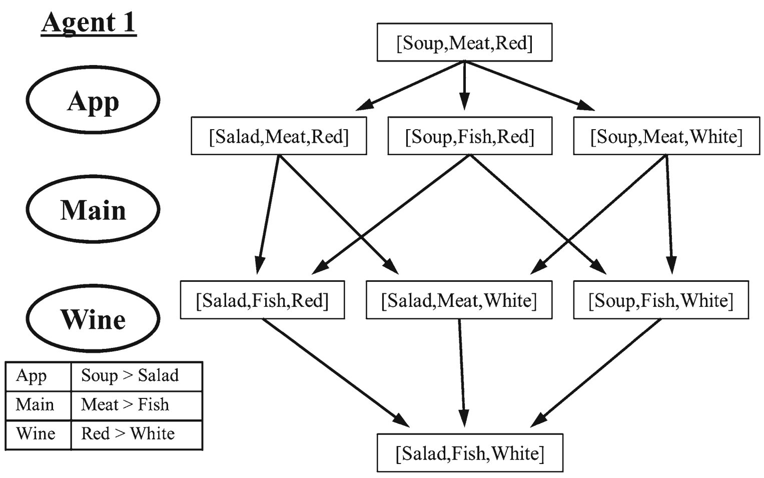
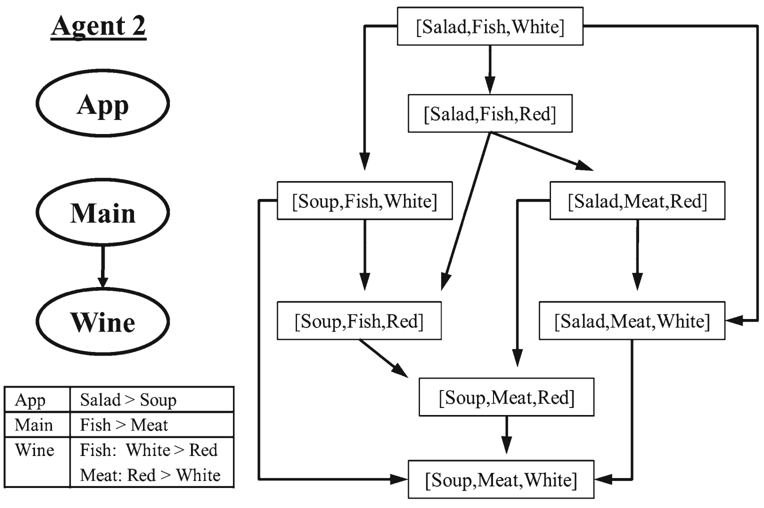


Fig. 2 A CP-net with the corresponding graph representing the partial order between all possible outcomes

order over the outcome ordering is needed, we need to linearize such incomparable pairs of outcomes [10, 14]. Throughout this paper we will use the same linearization for all our examples and proofs in order to be consistent. Given an outcome, we can associate with it a vector of as many elements as the number of variables, indicating how much the value for each variable is preferred in the context of the values for its parent variables. If there are just two values in the variable domains, this will be a binary vector, where 0 means most preferred and 1 means least preferred. For the example in Fig. 3, we associate the vector 010 to outcome {*Salad*, *Meat*, *Red*}, since *Salad* is the most preferred value for variable *App*, *Meat* is the least preferred value for variable *Main*, and, in the context of *Meat*, *Red* is the most preferred value for *Wine*. Of course the vector for the optimal outcome is 000 and it is 111 for the worst outcome.

Given an outcome, the next best outcome can then be found by just increasing by one the number represented by its vector and then passing to the corresponding outcome. For example, the next best outcome starting from {*Salad*, *Meat*, *White*}, whose vector is 011, is the outcome {*Soup*, *Fish*, *White*}, whose vector is 100. The intuition under this definition of linearization is that the next best outcome is obtained by decreasing the preference in one of the variables, according to a fixed variable ordering. In our example the variable ordering is *App*, then *Main*, then *Wine*. To generate the next best solution, we lower the preference of the first variable in the reverse ordering for which it is possible, and then we reset all subsequent variables to their most preferred value.

Example 3 In Fig. 3 we show the CP-net of agent 2 and its induced outcome ordering. Notice that {*Salad*, *Fish*, *Red*} precedes {*Soup*, *Fish*, *White*} in the partial order, since {*Salad*, *Fish*, *Red*} has a less preferred assignment in a dependent variable, which is less important than an independent variable (this is a consequence of the ceteris paribus semantics). If we were to linearize the partial order shown in



App	Salad > Soup
Main	Fish > Meat
Wine	Fish: White > Red Meat: Red > White

Fig. 3 A CP-net with the outcome ordering

Fig. 3 into a strict total order using a lexicographic linearization method over the variable names we would have: $\{Salad, Fish, White\} \succ \{Salad, Fish, Red\} \succ \{Salad, Meat, Red\} \succ \{Soup, Fish, White\} \succ \{Salad, Meat, White\} \succ \{Soup, Fish, Red\} \succ \{Soup, Meat, Red\} \succ \{Soup, Meat, White\}$.

Given a set of issues, domains, and some preferred outcome p, there is always a CP-net that has p as its optimal outcome [7]. However, given issues, domains, and an outcome ordering $p_1, \ldots, p_n$, there may be no CP-net whose induced outcome ordering coincides with the given ordering. In fact, CP-nets cannot model all outcome orderings. For example, two outcomes differing for the value of one issue must necessarily be ordered in the preorder induced by a CP-net. So, if we are given an ordering where two outcomes differ on a single variable that is incomparable, there is no CP-net that can induce this ordering. Although CP-nets have a limited expressiveness, they also present inherent advantages because of their ability to capture qualitative preferences, which may be convenient especially in the context of preference elicitation.

2.2 Voting

Voting theory [3] is a wide research area that considers scenarios (elections) where a collection of voters (or agents) vote by expressing their preferences over a set of candidates (or outcomes) and a voting rule decides the winner. Voting theory provides many voting rules to aggregate agents' preferences. Each rule takes, as input, a partial or complete preference ordering of the agents and gives, as output, a winner (an outcome that is best according to the rule). If there are only two candidates, the best rule, according to many criteria, is majority voting [2, 30]. When there are more than two candidates, there are many voting rules one could use (Plurality, Borda, STV, approval, etc.), each with its advantages and drawbacks.

In this paper we consider the following voting rules, modified slightly to work when agents express their preferences as CP-nets:

- Plurality: The candidate ranked first by a voter receives a point. The candidate with the most points wins the election. When there are two candidates, plurality coincides with majority.
- Veto: Each voter chooses a candidate to veto (disapprove) with all other candidates receiving a point. The candidate with the most points wins the election.
- k-approval: Each voter approves of k out of m candidates ($k \leq m$) and disapproves of the remaining candidates. All approved candidates receive one point. The candidate with the most points wins the election. Notice that Plurality and Veto are special cases of k-approval.

Voting theory provides an axiomatic characterization of voting rules in terms of desirable properties such as: the absence of a dictator, unanimity, anonymity, neutrality, monotonicity, independence of irrelevant alternatives, and resistance to manipulation and bribery. In this paper we focus on bribery; we point the reader to the book by Arrow et al. [3] for an introduction to the other facets, and to [2], a classical impossibility result about some of these properties.

Given a profile, a briber is an outside agent that wants to affect the outcome of an election by convincing some agents (voters) to change their votes so that a preferred candidate wins [21]. The briber can convince agents by paying them and is usually subject to a limitation of his budget. Later in the paper we formally define the kind of bribery problem we consider.

The bribery problem, first introduced by Faliszewski et al. [21], asks if an outside agent can affect the result of an election so that some preferred candidate p can win by changing the votes of some set of agents. Generally, the outside agent is subject to some resource bounds either on the total amount of change he or she can affect on any individual voter or on the total amount of change across all voters [18, 21]. Bribery problems in the setting where preferences are given as total linear orders is a well studied area in computational social choice. Bribery in the setting with total orders has a number of useful interpretations including campaign (resource) management [17, 18], as a metric for determining the margin of victory in an election [37], and as a metric for approximating strategy proofness [5]. Bribery has been studied under a variety of models including deterministic and probabilistic models [11, 19]; complete and imperfect information models [21]; and uniform and non-uniform prices [18, 20, 21]. Results from bribery scenarios that most closely match bribery in CP-nets are shown in Table 1.

Bribery is a closely related problem to manipulation. Manipulation occurs when a subset of the agents can get a better outcome by misreporting their preferences. More formally, in the *Constructive Coalitional Manipulation (CCM)* problem [13] we are given a set of agents X with fixed votes, a set Y of agents with unspecified votes, and a candidate $p \in C$ where C is the set of possible candidates. We ask if there is an assignment of votes to the agents of Y such that running the election on the profile $X \cup Y$ results in p winning the election. Certain forms of the manipulation problem can therefore be seen as bribery problems where the agents which can modify their vote are fixed and the budget is unlimited.

While every reasonable voting rule is manipulable by the Gibbard–Satterthwaite Theorem [23, 36], a rule may be said to be resistant to manipulation if it is computationally difficult to decide how to manipulate. This has led to intensive studies of the computational properties of bribery and manipulation for a variety of voting rules with complete and incomplete preferences [13, 21, 25, 26, 33–35, 39] and with voters preferences represented by CP-nets [12, 38].

2.3 Voting with CP-nets

We consider a set of n agents with preferences over a common set of candidates with a combinatorial structure: there is a common set of m binary issues and the set of candidates is the Cartesian product of their domains. Each candidate (or outcome) is an assignment of values to all issues, thus we have 2^m candidates.

Table 1 Complexity results for oher versions of the bribery problem

	Basic bribery	Swap bribery
Plurality	P	P
Veto	P	P
k-approval	P	NP-c ($k \geq 2$)

Results in the first column are from Faliszewski et al. [21] and the results for the second column are from Elkind et al. [18]

Each agent expresses his preferences over the candidates via a compact acyclic CP-net. Moreover, when we use a sequential approach to voting, there is a total ordering O over the issues such that, in each CP-net, each issue must be independent from all issues following it in the ordering O.

A *profile* (P, O) is a collection P of n CP-nets over m common issues and a total ordering O over the issues that satisfies the above property. This is called an O-legal profile in [24]. Notice that the CP-nets appearing in such profiles do not necessarily have the same dependency graphs.

To define a voting scheme over O-legal profiles Lang [24] proposes a sequential approach where, at each step, agents vote over a single issue by using a chosen voting rule. This voting scheme is based on a total order over the issues that is compatible with the (acyclic) dependency graphs of the CP-nets: in each CP-net, each issue must be independent from all issues following it in the ordering. After each vote, a value is selected for the considered issue, and this choice is returned to all the agents who include this information in their CP-net. After as many steps as the number of issues, the winner outcome is built by collecting all the winning values for each issue. In this paper we consider a sequential approach but we also examine a one-step approach which computes the winner by considering all the CP-nets at once.

3 Bribery problem definition

The bribery problem we consider has three parameters: the way a winner is chosen from the given profile, the bribery actions, and the cost scheme for the briber's requests. In the following we precisely define these three items and the bribery problem for combinatorial domains.

3.1 Winner determination

To determine the winner outcome, we consider four approaches: sequential majority (SM), one-step plurality (OP), one-step veto (OV), and one-step k-approval (OK). In addition to the sequential approach there is a one-step approach, where each agent votes over decisions regarding all variables, not just one at a time. For all the voting rules we consider (SM, OP, OV, and OK), it is computationally easy for an agent to compute his vote. An approach like OP is however less satisfactory than the sequential approaches in terms of ballot expressiveness: since the number of solutions is exponentially large with respect to the number of candidates, there is an exponential number of solutions which are not voted by any agent. However, if we want agents to be able to compute their vote in polynomial time, we need to set a bound on the number of solutions they can vote for, and this means that in all cases an exponentially large number of solutions will not be voted. So there is a trade-off between easiness of computing votes and ballot expressiveness.

3.1.1 Sequential majority (SM)

Sequential Majority is Lang's sequential procedure [24] instantiated on binary issues. For this approach, we consider O-legal profiles, and we vote for each issue in the ordering O using majority (since issues are binary). The value chosen for an issue is returned to all agents who then fix the issue to that value in their respective CP-nets. When voting has been completed on all issues, the winner outcome is defined by the issue instantiations chosen by majority at each of the voting steps.

Example 4 Consider our running example of three agents illustrated in Fig. 1. Since we have three issues we will hold three separate sub-elections, all decided by majority. Since $O = (App, Main, Wine)$ for this example, we first take a majority vote for App: *Salad* is preferred by a vote to *Soup*. We then return this value to all agents and have them fix this value in their CP-net. Moving to the next issue, we have that *Fish* is preferred to *Meat* by a two to one majority. The partial assignment for all agents is now $\{Salad, Fish\}$. For $Wine$, we have that, given this partial assignment, two out of three agents vote for *White*. Thus the winning outcome for this example, with the SM procedure, is $\{Salad, Fish, White\}$.

3.1.2 One-step plurality (OP)

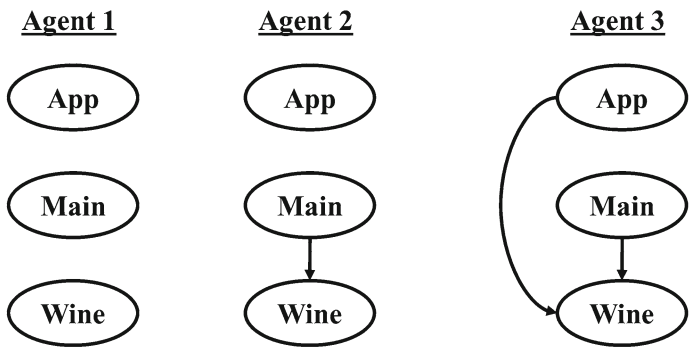
In One-step Plurality we ask each agent to provide their optimal outcome according to their CP-net. We then use the plurality rule to evaluate the winning outcome. Note that for this method (and all the other one-step methods) we do not require the O-legality constraint, but only that agents are able to compute their top outcome.

Example 5 Consider our running example shown in Fig. 1. Agent 1 casts his vote for $\{Soup, Meat, Red\}$, agent 2 for $\{Salad, Fish, White\}$, and agent 3 for $\{Salad, Fish, White\}$. Collecting these votes under the plurality rule, the winning outcome is $\{Salad, Fish, White\}$.

3.1.3 One-step veto (OV)

In One-step Veto we ask each agent to provide their worst outcome according to their CP-net. Each agent can determine their worst outcome by flipping all of their individual cp-statements, the reversed cp-statements for our running example are illustrated in Fig. 4. For this method there may be many candidates in the winner set because the number of alternatives may be exponential in the number of voters. Moreover, since we are using this as a social choice procedure, we only need to select an alternative from the winning set which we can do in polynomial time.

Example 6 Consider our running example shown in Fig. 1. First, each agent must reverse all the cp-statements in their CP-nets. This will leave the agents with the CP-nets illustrated in Fig. 4. We see that agent 1 vetoes $\{Salad, Fish, White\}$, while agent 2 and agent 3 both veto $\{Soup, Meat, White\}$. In this case we have vetos for two of the $2^3 = 8$ possible outcomes. Therefore, the winning set is made up of six alternatives (ordered by the variable ordering and then lexicographically): $\{Salad, Fish, Red\}$, $\{Salad, Meat, White\}$, $\{Salad, Meat, Red\}$, $\{Soup, Fish, Red\}$, $\{Soup, Fish, White\}$, $\{Soup, Meat, Red\}$. Here we would select $\{Salad, Fish, Red\}$ if we employed a lexicographic tie-breaking rule.



App	Salad > Soup
Main	Fish > Meat
Wine	White > Red

App	Soup > Salad
Main	Meat > Fish
Wine	Fish: Red > White Meat: White > Red

App	Soup > Salad
Main	Meat > Fish
Wine	Salad, Fish: Red > White Salad, Meat: White > Red Soup, Meat: White > Red Soup, Fish: White > Red

Fig. 4 The three "reversed" CP-nets corresponding to the three CP-nets shown in Fig. 1

3.1.4 One-step k-approval (OK)

In one-step k-approval we ask each agent to provide their top k outcomes according to their CP-net and we use k-approval to decide which of these outcomes is the winner.

Example 7 Consider the situation where $k = 2$ in our running example illustrated in Fig. 1. The ordering of the preferences of agent 1 can be seen in Fig. 2. We see that the top outcome for agent 1 is $\{Soup, Meat, Red\}$. In order for agent 1 to have two approved outcomes we have to linearize the preferences in some way (since there are three options that are incomparable). We choose to linearize them by the reverse ordering of the variables, thus a flip from $\{Red\}$ to $\{White\}$ brings us to the next best outcome. Therefore, agent 1 approves of $\{Soup, Meat, Red\}$ and $\{Soup, Meat, White\}$.

We can see the ordering of assignments for agent 2 in Fig. 3. With the same reasoning as for agent 1, agent 2 approves $\{Salad, Fish, White\}$ and $\{Salad, Fish, Red\}$. Agent 3 has instead two ordered outcomes that are more preferred than all the other outcomes ($\{Salad, Fish, White\}$ and $\{Salad, Fish, Red\}$), so they will be approved.

3.2 Bribery actions

The bribery actions are the changes agents perform in their preferences in response to a briber's request. In most bribery frameworks agents express their preferences over the candidates via a total order and the briber asks agents to make changes within this total order. In our setting, agents have a CP-net instead of an explicit outcome ordering. Thus, we choose to define the bribery actions as changes in the total order of some conditional preference statements. In this paper we consider binary issues, so changing a cp-statement means flipping the positions of the two values of an issue.

In a CP-net, a cp-statement is associated with a certain issue and issues are of two kinds: independent and dependent. Independent issues have indegree zero in the dependency graph while dependent issues have indegree ≥ 1. Independent variables can cause greater changes in the final preference order and, since they are independent, agents have "pure preferences" over them, independent from any other assignment. Therefore, we consider independent variables to be more important than dependent variables.

We distinguish bribery actions by specifying in which cp-statements the briber is allowed to request a change:

- IV: The briber can only request a change in the cp-statements of independent variables.
- DV: The briber can only request a change in the cp-statement of dependent variables.
- IV+DV: The briber can ask for any change in any cp-statement.

Notice that, with any of these bribery actions, no dependency can be added in the CP-nets. In fact, the outside actor can only remove dependencies through the variable swaps. Thus, the new CP-nets are still compatible with one another and with the ordering O over the issues. This is only important for SM. Additionally, if only one or the other type of bribery actions is available, then there could be outcomes that cannot become the most preferred outcome, as illustrated in the following example.

Example 8 Consider our running example shown in Fig. 1. Suppose the briber's preferred candidate is $p = \{Soup, Meat, White\}$ and we are using the OP voting rule. Thus he needs to bribe some of the agents to have p as their top outcome.

When considering only IV bribes, all the preferences of agent 1 can be changed, since the issues are all independent. For agent 1, we only need to bribe him to swap his assignment such that $Wine = White$; p now has one vote. For agent 2, both *App* and *Main* are independent variables and we can bribe the agent on these two variables only. When we bribe these variables (swapping them to make the variable assignments match p) we see that the most preferred assignment of agent 2 is $Wine = Red$, given $Main = Meat$. Since we are only allowing IV bribes, we cannot make agent 2 vote for p. For agent 3 we have the same problem as for agent 2. Though we can set his preferences to $App = Soup$ and $Main = Meat$, this causes $Wine = Red$ in the CP-net. Thus we cannot make p the winner of the election.

Consider the case of only DV bribes. In this case we cannot change any of the preferences of agent 1 because all his issues are independent. Furthermore, we cannot bribe agent 2 or agent 3 to change his preferences for *App*, because no agent has *App* as a dependent variable. In this case we could change the assignment of *Wine* but this is not enough to make p a winner.

In the case of IV+DV bribes, we can make p a winner. To do this, we perform the same bribes to the agents as we did in the IV example. However, we can also bribe agent 2 and agent 3 to have $Wine = White$. Using IV+DV we can make all agents have p as their top preference and thus a winner in the election.

3.3 Cost schemes

In traditional bribery domains, the cost of the bribery actions is either fixed for any amount of change for each agent [21], variable per agent for any amount of change [21], or it depends on the number of swaps to be performed on the current total order [18, 20]. However, we feel that none of these methods captures an effective cost scheme when agents express their preferences as CP-nets and voting is done over a combinatorial domain. We do not want to work on the outcome ordering, since it can be exponentially large; we would rather modify the CP-net directly. However, a single flip in a CP-net preference statement can lead to a large change in the outcome ordering and we need to take this into account. Therefore, in this paper we consider five different cost schemes. Some of these cost schemes closely resemble the costs in more traditional bribery domains while others are more closely tied to the use of the CP-net formalism and applicable to combinatorial domains.

The five cost schemes we investigate in this paper are listed here from the least expressive to the most expressive:

- C_{EQUAL}: A unit cost allows any number of flips in the cp-statements of a CP-net. This method is the same as the one proposed by Faliszewski et al. in the original definition of bribery [21].
- C_{FLIP}: The cost of changing a CP-net is the total number of individual cp-statements that must be flipped in order to change one profile P_i to a target profile P_t. This method captures a notion of how much change the briber is asking for, and makes the cost of bribery proportional to the total amount of change. This method is similar to the swap bribery method introduced by Elkind et al. [18] applied to CP-nets instead of strict total orders.
- C_{LEVEL}: We expect an issue that is closer to being independent is more important, as in some other structured preference formalisms [6], than an issue deeper in a dependency graph. Therefore, we expect that the cost of changing a more influential issue should be higher than lower level issues. We link the cost of a flip to this importance: the cost of changing a CP-net is the total number of flips performed in the cp-statements, each weighted according to the level of the relevant issue (in the original CP-net configuration). We define the *level* of an issue recursively as:

$$level(x) = \begin{cases} 1 & \text{if } x \text{ is an independent issue;} \\ i+1 & \text{if all parents of } x \text{ are in levels } \{1, \ldots, i\} \\ & \text{and there is a parent in level } i. \end{cases} \quad (1)$$

  We can then precisely define the cost as $\sum_x flip(x) \cdot (k + 1 - level(x))$, where x ranges over the issues, k is the number of levels in the CP-net, and $flip(x)$ is the number of flips performed in cp-statements associated with x.
- C_{ANY}: The total cost is the number of flips, each weighted by a specific cost. Formally, we define a vector for each agent that, for every variable $v_i \in M$, maps v_i to a value of $\mathbb{N}$ so that modifying any variable in the instance has its own associated price. This is similar to the nonuniform bribery model introduced by Faliszewski [18, 20].
- C_{DIST}: We assume that each agent provides a total order over the issues, which is compatible with the dependency graph of the CP-net. This allows the agent to

univocally define a linearization of the outcome ordering [8]. The bribery cost associated to a request to vote for a certain candidate p is the distance (i.e., the number of elements) in such a linearization between the current optimal outcome and p.

Each of the above cost schemes can be generalized to account for agents with different bribing costs by associating a certain cost to each agent. This allows each agent to have its own cost function. We assume, where necessary, that each agent has their own individual bribing cost and we are provided a vector $\mathbf{Q} \in (\mathbb{N})^n$ where each $Q[i]$ denotes the individual bribing cost of agent i. We multiply this bribing cost by the cost returned from the given cost scheme. This creates a situation similar to plurality-\$bribery introduced by Faliszewski et al. [21].

In many variations of the bribery and manipulation problems, it is standard to attach weights to each voter [13, 21]. Weights allow for modelling settings where voters have differing levels of importance, such as shareholder votes in a company. We will sometimes assume a weight vector is provided $\mathbf{W} \in (\mathbb{N})^n$ where $W[i]$ represents the weight associated to voter i.

It is important to note that the distance of an outcome from the optimal one is not necessarily linked to the number of flips needed in the cp-statements to make that outcome optimal, as the following example shows.

Example 9 Consider the CP-net and the outcome ordering of agent 2 in Fig. 3. The optimal outcome is $\{Salad, Fish, White\}$, and the outcome $\{Salad, Meat, Red\}$ dominates $\{Salad, Meat, White\}$, therefore $\{Salad, Meat, White\}$ is farther from the optimal outcome than the outcome $\{Salad, Meat, Red\}$ from the optimal outcome in the partial order. To make $\{Salad, Meat, White\}$ the optimal outcome, we only need to flip one cp-statement ($Fish \rightarrow Meat$). Instead, to make $\{Salad, Meat, Red\}$ the optimal outcome, we need two flips ($Fish \rightarrow Meat$ and $White \rightarrow Red$) for a total cost of 2 for C_{FLIP} (versus 1 for the other outcome). For C_{LEVEL}, we have a similar situation: changing the optimal outcome to $\{Salad, Meat, White\}$ has cost 1 while switching to $\{Salad, Meat, Red\}$ has cost $2 \cdot 1 + 1 = 3$ (one independent and one dependent flip). Therefore, the cost schemes C_{LEVEL} and C_{FLIP} (and the more general C_{ANY}) do not follow the principle that the cost should be proportional to the distance from the optimal outcome.

3.4 Combinatorial bribery

We are now ready to formally state the bribery problem we study in this paper. We can state this problem for each choice of winner determination rule $D \in \{SM, OP, OK, OV\}$, bribery action $A \in \{IV, DV, IV + DV\}$, and cost scheme $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}, C_{\text{DIST}}\}$.

Name:	(D,A,C)-Bribery
Given:	A profile (P, O), where P is a collection of n compact, acyclic CP-nets with m binary issues and O is a total ordering of the issues (when necessary), a budget $B \in \mathbb{N}$, a preferred outcome p, and (when necessary) a bribing cost vector $\mathbf{Q} \in (\mathbb{N})^n$, and weights vector $\mathbf{W} \in (\mathbb{N})^n$.

Question: Is there a way for an outside actor to make p win in profile (P, O) with winner determination rule $D \in \{SM, OP, OK, OV\}$ using bribery actions $A \in \{IV, DV, IV + DV\}$, paying according to the cost scheme $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}, C_{\text{DIST}}\}$ and bribing cost vector $\mathbf{Q}$, with weights vector $\mathbf{W}$, and without exceeding budget B?

We assume, as in other works on bribery such as those by Elkind et al. [18] and Faliszewski et al. [21], a non-unique winner model: we are only asked to elevate p into the winning set. Some of our reductions can be extended to a unique winner model but we focus on the non-unique winner model. While we obtain mostly easiness results under this winner model (as did Bartholdi et al. [4]), a complete investigation of tie-breaking and its effect on the complexity of manipulation is outside the scope of this paper. In fact, a careful study of tie-breaking mechanisms may be quite complex as shown by Obraztsova et al. [31, 32] among others [22] and requires its own course of study.

Example 10 Given the CP-nets shown in Fig. 1 and $O = (App, Main, Wine)$ we can consider several examples of our bribery problem parameterized by different attributes.

- (SM-IV-C_{EQUAL})-Bribery: Consider $B = 10$, $\mathbf{Q} = [5, 5, 5]$, and $p = \{Soup, Meat, Red\}$. We need to select some bribes, only in the IV variables, in order to make p a winner of the election decided by SM. Agent 1 has already p as its top outcome, so we do not need to bribe him at all. If we give bribes to either agent 2 or agent 3 for App and $Main$, they will change $Wine \to Red$. Since we are using C_{EQUAL} and $Q[2] = Q[3] = 5$ we can make both assignment changes in the cp-statement of either agent for a cost of 5. This makes each individual element of p a majority winner at its respective level. Therefore this is a “yes” instance of our problem.
- (OV-DV-C_{FLIP})-Bribery: Consider $B = 5$, $\mathbf{Q} = [1, 1, 2]$, and $p = \{Salad, Fish, White\}$. Notice that we are using OV and we have three agents and $2^3 = 8$ possible outcomes. In this instance there will necessarily be some outcomes that do not have any vetoes. Therefore, we must make sure that p has no vetoes since the winner set will contain only outcomes that have no vetoes. In this case we are limited to only DV bribes. This means we cannot bribe agent 1 as he has no dependent variables. Unfortunately, agent 1 is the only one vetoing p and we can’t change his mind. Therefore, this is a “no” instance of our problem.
- (OP-{IV+DV}-C_{LEVEL})-Bribery: Consider an instance with $B = 3$, $\mathbf{Q} = [2, 1, 2]$, and $p = \{Salad, Meat, White\}$. No agent has p as their top preference and thus p is not winning. Notice that, in order to convince agent 1 to have p as his top outcome, it requires two bribes to independent variables. Since all variables for agent 1 are at level one of its CP-nets, the cost of this bribe is $Q[1] \cdot 2 = 4$. However, bribing either agent 2 or agent 3 requires a bribe at level one and a bribe at level two, therefore the cost of the bribe for agent 2 is $Q[2] \cdot (1 \cdot 1 + 1 \cdot 2) = 3$ while the cost of bribing agent 3 is $Q[3] \cdot (1 \cdot 1 + 1 \cdot 2) = 6$. We choose to bribe agent 2 to have p as his top candidate. This means that we have one vote each for p, $\{Soup, Meat, Red\}$, and $\{Salad, Fish, White\}$. We have elevated p into the winning set and spent no more than our budget, so this is a “yes” instance.

4 Winner determination and vote modification

The computational complexity of the bribery problem is interesting only when it is easy to determine the winner of an election, since otherwise bribery would be trivially hard. Winner determination is easy to compute for all approaches we consider.

Theorem 1 *Winner determination is in* P *for SM, OP, OV, and OK (when k is bounded).*

Proof Sequential Majority (SM) works issue-by-issue, and at each issue it applies the majority voting rule. As there are a limited number of issues, the overall procedure is in P.

For One-step Plurality (OP), we need to compute the optimal outcome for each CP-net, which is polynomial-time computable since we are dealing with acyclic CP-nets. Then, plurality is applied, and this again is a polynomial-time computable step.

The winner according to One-step Veto (OV) can be achieved by reversing each total order in all cp-statements of the CP-nets. In fact, by doing this, the optimal outcome of the new CP-net is the worst outcome of the original one. Notice that in an acyclic CP-net we have exactly one worst outcome. This reversal step takes polynomial time in the size of the CP-nets since they are compact. After the reversal step, we just choose an outcome which appears the smallest number of times (possibly zero) as the optimal outcome of the reversed CP-nets. This again takes polynomial time with respect to the size of the input.

One-step k-approval (OK) needs to get the top k outcomes from each CP-net. If k is bounded, this is easy since the CP-nets are acyclic [8]. Therefore, providing the ballots is polynomial, as is applying k-approval. □

In a bribery scenario, the briber has a desired candidate, say p. To make p the overall winner, the briber may ask some agents to change their vote to a vote for p or for another candidate, paying for this change according to the cost scheme. Computing the cost of a briber's request is easy in our scenario, no matter which bribery actions or cost schemes we use.

Theorem 2 *Given a CP-net and an outcome p, determining if the CP-net can be changed to make p its optimal outcome, and, if so, determining the minimum cost to perform such a change, can be computed in polynomial time if we use the cost schemes* C_{EQUAL}, C_{FLIP}, C_{LEVEL}, *or* C_{ANY}.

Proof The proof mirrors the proof for Proposition 3.2 in Elkind et al. [18]. The same argument works for IV, DV, or both. Assume we can use all bribery actions, thus $IV + DV$. Then, any CP-net can be changed to vote for p (that is, to make p its optimal outcome).

For C_{EQUAL}, it is trivial to compute the cost, as it does not depend on the amount of changes to the CP-net: if p is already the top outcome of the CP-net, the cost is 0, otherwise it is 1.

For C_{FLIP}, we need to compute the minimum number of flips to be done on the CP-net. There are some flips that are necessary to make p the top outcome. In order to find them, starting from the independent issues, we flip the cp-statements which do not have the corresponding value of p as their preferred value. We then proceed to each dependent variable, as soon as its parents are processed, and do the same in its relevant cp-statement. At the end, p is the top outcome in the resulting CP-net, and the number of flips performed is the minimum cost to make p win.

For C_{LEVEL} or C_{ANY}: the flips are the same as above but the cost takes into account the levels or the cost of each flip.

Notice that this algorithm gives us the minimum set of flips, but the same result could be achieved also by performing other useless flips. However, what we are interested in is computing the minimum cost to change the CP-net.

With only IV or DV, as soon as a necessary flip cannot be done, it means that p cannot be made the optimal outcome. Therefore, in polynomial time, we can compute the cost to change any vote to any other vote or discover that it is not possible. □

With the cost scheme C_{DIST}, the cost of a vote change is not related to the flips to be performed in the CP-net, but to the distance between the current optimal outcome and the outcome to vote for. This distance may be exponentially large with respect to the size of the CP-net. However, the computation of such a cost is always polynomial.

Theorem 3 *Given a CP-net and an outcome p, determining the cost of voting for p, according to the cost scheme C_{DIST}, takes polynomial time. It is also easy to know which modifications to make in the CP-net to make p the optimal outcome.*

Proof What we need to do is to compute the Boolean vector associated to outcome p, that tells us how much worse p is with respect to the optimal outcome [8]. More precisely, given an outcome over n issues, we build a Boolean vector of length n such that, for each issue i, we set the bit i of the vector equal to 0 if the corresponding value in the outcome is preferred, given the values of its parent issues in the CP-net, otherwise we set its value to 1. Obviously, the best outcome will be associated to a Boolean vector with all 0's, while the worst outcome will have a Boolean vector with all 1's. Once we have the Boolean vector associated to outcome p, we compute its decimal value. This is exactly the distance from the optimal outcome in the linearization of the outcome ordering as defined in [8]. This reasoning holds no matter which bribery actions are allowed. To understand which flips to make to vote for p, the voter can then follow the same approach as in the proof of Theorem 2. □

5 Bribery and sequential majority

In this section we consider the computational complexity of bribery when we determine the winner via sequential majority.

Theorem 4 *(SM,IV,C)-Bribery, (SM,DV,C)-Bribery, and (SM,IV+DV,C)-Bribery are in* P *when $C \in \{C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}\}$.*

Proof We show a polynomial time algorithm for solving the bribery problem in an instance of $(SM, IV + DV, C_{\text{FLIP}})$-Bribery. The argument is independent of the selection of C and the same algorithm can be used for DV and IV bribery actions. We can determine whether or not the outside agent can achieve its agenda in polynomial time. We provide an algorithm to determine the cost of the optimal sequence of bribes.

Given a profile (P, O), we consider the issues in the order O. For the first issue, we compute the minimum number of flips to be performed in the CP-nets in order to achieve a majority for the same value that appears in p for that issue. Since we are using SM, we can select the agents to bribe by starting from the cheapest ones (according to $\mathbf{Q}$), until we achieve a majority for the considered issue. In fact, using SM, the resulting preferred value for this issue will be set in all agents' CP-nets (not just those where this value was most preferred).

For each of the other issues in O we perform the same computation. We can ignore any conditional dependencies; as long as we achieve our agenda at this level, our bribery actions on a per agent basis will have no effect on the cost of changing an agent's preference for the next issue in O. This is because we must pass every issue at every level and passing an issue sets its value in *all* agents' CP-nets. Therefore, all effects farther down in the variable ordering are the same for all agents and we can ignore them.

When all issues have been considered, the number of performed flips required to get the correct majority for each issue is the cost of bribing so that p wins. The total cost is within the budget $\leq B$ if and only if the bribery problem has a solution.

If we use the cost scheme C_{ANY} the choice of the agents to bribe for each issue still only requires us to greedily select the cheapest agent at each level.

For C_{LEVEL} we must consider the level of the variables when computing the cost for each agent. However, since the topological order of any agent's preference dependencies must respect O we can traverse the variables in the order described in O. This will ensure that we consider the variables in cost order for each agent and that we do not change a variable before processing all variables it depends upon. Therefore, we can greedily select the cheapest agents to bribe at every level with C_{LEVEL}.

A very similar algorithm can be used for IV or DV by giving a cost of $B + 1$ to an agent's bribery prices if p cannot be made its best outcome given the restrictions. □

The above theorem does not consider the cost scheme C_{EQUAL}. With C_{EQUAL}, the cost for changing a CP-net is always 1, no matter how many flips are needed. Therefore, the decision of which CP-nets to change for a particular issue is strictly related to this same decision for another issue: if we change the same CP-nets, the cost will not increase. So, to compute the minimum bribery cost, the computation we do for one issue depends on the computation for a different issue. This makes the bribery problem computationally hard, as the following theorem formally shows.

Theorem 5 *$(SM, IV, C_{\text{EQUAL}})$-Bribery, $(SM, DV, C_{\text{EQUAL}})$-Bribery, and $(SM, IV + DV, C_{\text{EQUAL}})$-Bribery are* NP-*complete.*

Proof Membership in NP is easy to prove, since it is an immediate consequence of Theorem 1 and a guess and check algorithm. In order to show completeness, we provide a polynomial-time reduction from the Optimal Lobbying (OL) problem [11]. In this problem, we are given an $n \times m$ 0/1 matrix E and a 0/1 vector $\mathbf{x}$ of length m where each column of E represents an issue and each row of E represents a voter.

We say E is a binary approval matrix with 1 corresponding to a "yes" vote and $\mathbf{x}$ is the target group decision. We then ask if there is a choice of k rows of the matrix E such that these rows can be edited so that the majority of votes in each column matches the target vector $\mathbf{x}$. This problem is shown to be $W[2]$-complete [16] via a reduction from k-Dominating Set [11]. By giving a polynomial-time reduction from OL to our bribery problems, we show that our problems are NP-complete and, indeed, even $W[2]$-hard with parameter B.

We start with $(SM, IV, C_{\text{EQUAL}})$-Bribery. Given an instance $(E, \mathbf{x}, k)$ of OL, we construct an instance of $(SM, IV, C_{\text{EQUAL}})$-Bribery containing CP-nets with only independent variables. The number of issues, m, is equal to the number of columns in E. For each row of E, we create a voter with the preferences over the m variables as described in the row of E. Finally, we set the price of bribery for each voter to be 1, the budget $B = k$, the weights of the agents all equal, and the preferred outcome $p = \mathbf{x}$. With this mapping, outcome p wins the election if and only if there is a selection of k rows of E such that $\mathbf{x}$ becomes the winning agenda of the OL instance. Therefore $(SM, IV, C_{\text{EQUAL}})$-Bribery is NP-complete.

The same reduction works also for IV+DV. For DV, we choose an instance of the bribery problem as above, except that there is one more issue and each voter has an additional independent variable on which all other variables depend. The preference of all voters on the new variable is $1 > 0$. For the other variables, the corresponding row of E will provide the preference associated to the value 1 of the new variable, while for the value 0 we give the bitwise complementary preference. The last difference is that now p is $1\mathbf{x}$. With this mapping, p wins the election if and only if the given OL instance has a positive answer. □

With weighted agents, the winner of the election will be computed by a weighted majority for each issue. We prove that, in this case, bribery with all cost schemes is computationally difficult. We denote by weighted-X the bribery problem X with weighted agents.

Faliszewski et al. [21] show NP-completeness of a problem called plurality-weighted-\$bribery. In such a problem each candidate c_i is associated with a price $\$_i \in \mathbb{N}$ and each voter v_i has a weight $w_i \in \mathbb{N}$, and the voting rule used is plurality. The problem is to determine if a favorite candidate, say p, can be made a winner by bribing voters within a budget B. In fact, Faliszewski et al. show this problem to be NP-complete for just two candidates. The following theorem can be proven via polynomial reductions from plurality-weighted-\$bribery over two candidates to our problems.

Theorem 6 *Weighted-(SM,A,C)-Bribery where* $A \in \{IV, DV, IV + DV\}$ *and where* $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}, C_{\text{DIST}}\}$ *is* NP-*complete.*

Proof Membership in NP is an immediate consequence of Theorem 1 and a guess and check algorithm. We can reduce the plurality-weighted-\$bribery problem over two candidates to an instance of our problem where we have the same number of voters, the same costs and weights, and the same budget as in the original problem. Moreover, there is only one issue, thus the voters have a single independent variable with two values corresponding to the two candidates, one of which corresponds to the briber's preferred candidate p. Since this reduction uses only one independent variable, it works also for IV+DV, as well as for all other cost schemes. The reduction holds for DV since it is enough to add an independent variable with preferences $1 > 0$ by all voters, on which the other variables depend, with preferences $1 : p > \bar{p}$. Now, $1p$ is the new briber's preferred candidate. □

However, if we use the cost scheme C_{FLIP} and we assume that the bribing costs are the same for all voters, it is possible to show that the bribery problem becomes easy (the proof is similar to that of Theorem 4).

Theorem 7 *Weighted-(SM,IV,C_{FLIP})-Bribery, weighted-(SM,DV,C_{FLIP})-Bribery, and weighted-(SM,IV+DV,C_{FLIP})-Bribery are in* P *when the bribing costs in* **Q** *are all the same.*

Proof We can use the same argument as in the proof of Theorem 4. Since voters are weighted, we only need to achieve a weighted majority for each individual issue. Since all the bribing costs at each level are the same with the C_{FLIP} cost scheme, the cost to bribe any voter at any single level is equal for all voters. We can therefore bribe the voters from heaviest to lightest (in terms of their weight). This will be an optimal sequence of bribes at each individual level and will lead to the optimal bribery scheme overall. This is true no matter which bribery actions (IV, DV, or IV+DV) are allowed, since using IV or DV just restricts the issues over which it is possible to perform a flip in the CP-nets. □

6 Bribery and one-step plurality

Faliszewski [20] shows that plurality bribery in single issue elections with nonuniform cost functions is in P through the use of flow networks. The algorithm requires the enumeration of all possible elements of the candidate set as part of the construction of the flow network. In our model, the number of candidates can be exponential in the size of the input, so we cannot use that construction directly. However, we show that a similar technique can be used while not requiring the enumeration of an exponential number of candidates.

Theorem 8 *(OP,A,C)-Bribery, where $A \in \{IV, DV, IV + DV\}$ and $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{DIST}}\}$, is in* P.

Proof If the number of candidates, which is 2^m, is polynomial in the number of voters (n), we can directly use the construction in [20]. Therefore, we assume to have a number of candidates that is superpolynomial in the number of voters. We will show that a similar technique can be used in this case as well while assuring a polynomial-time algorithm.

We consider all $r \in \{1, \ldots, n\}$ and ask if the briber's preferred candidate p can be made a winner with exactly r votes without exceeding the budget B. If there is at least one r such that this is possible, then we accept, otherwise we reject. We show that, for each r, the corresponding decision problem can be solved in polynomial time. This means that the overall bribery problem is in P.

To solve the decision problem for a certain r, we transform this problem to a minimum-cost flow problem [1]. The network has a source s, a sink t, and three "layers" of nodes.

The first layer of nodes has one node for each voter $v_1, \ldots, v_n$. There are also n edges (s, v_i), each with capacity 1 and cost 0.

For the second layer of nodes, for each voter in the given profile, we include in this second layer nodes corresponding to p, to all the candidates with at least one vote (at most n), and to the n (non-voted) cheapest candidates for this voter, according to the cost scheme, for a total of at most $2n + 1$ candidates in the second layer. Intuitively, this second layer models the profile modified by the bribery actions, where each voter has either changed his vote or he has maintained the original vote. The important point is that the other non-voted candidates that we do not include in the second layer would never be used in the new profile obtained by bribing, since they cost too much. Providing n non-voted candidates for each voter is enough, since there are n voters and in the worst case each of them votes for a different candidate.

For each node x in the second layer corresponding to voter v_i, we add an edge from v_i to x, with capacity $+\infty$ and cost equal to the cost to bribe v_i to vote for the candidate corresponding to node x.

Finding the candidates corresponding to the nodes of the second layer, and the cost for the voter to vote for them, takes polynomial time, irrespective of the cost scheme. In fact, finding the voted candidates is easy since finding the optimal outcome in acyclic CP-nets takes polynomial time. For C_{DIST}, to find n cheapest non-voted candidates, we can just start from the optimal outcome of each CP-net and move down in a linearization of the outcome ordering. This takes polynomial time as shown in [8]. The same procedure can also be used for C_{EQUAL}, although all non-voted candidates have the same cost. For C_{FLIP}, we first consider the candidates which can be obtained by performing one flip in the CP-net and then computing the new optimal outcome. Then we proceed by increasing the number of flips, until we collect n non-voted candidates.

Given a voter, computing the cost for such a voter to vote for one of the collected candidates is easy for the non-voted candidates, since they are obtained by using the cost schemes (except for C_{EQUAL} where the cost is always 1). For the voted candidates, included the briber's preferred candidate p, it is easy to compute the cost if we use C_{FLIP} since we just need to count the number of necessary flips to make such a candidate win (see the proof of Theorem 2). For C_{EQUAL}, it is obviously easy since it is always 1 (or 0 if the candidate is already voted for by that agent). For C_{DIST}, we just need to compute the Boolean vector associated to the candidate and then take its decimal value, as described in the proof of Theorem 3.

In the third layer of the network, we include a node for each candidate who already appears somewhere in the network (up to $2n^2 + n$). One of these nodes represents the briber's preferred candidate p. These third layer nodes are the nodes that enforce the constraint that every candidate besides p cannot receive more than r votes, so that p can win, since we are considering a non-unique winner model. These nodes have an edge from the nodes of the second layer representing the same candidate, with zero cost and infinite capacity. The output link from each of the third layer nodes to the sink has capacity r. The cost is 0 for the edge from p to the sink, while for all other candidates it is a large integer M to force as much flow through the node representing candidate p as possible.

If we had included nodes for all the candidates in the second layer, we would have used a network equivalent to the one used in the proof of Theorem 3.1 in [20], which shows that there is a minimum cost flow of value n if and only if there is a way to solve the bribery problem. However, since we have a number of candidates which is superpolynomial in the size of the input, we would not have a polynomial-time algorithm. By including only the cheapest n alternative candidates for each voter, along with p and all the voted candidates, the result still holds. In fact, assume there is a minimal flow in the larger network which goes through one of the nodes which we omit. This means that a voter has been forced to vote for another, more expensive, non-voted candidate since all his cheapest candidates had already r votes each. However, this is not possible, since we have only a total of $n - 1$ votes that can be given by the other voters, and we provide n non-voted candidates.

We will build, at worst, n networks with $O(n^2)$ nodes and edges. Since the min cost max flow problem can be solved in polynomial time, the overall running time of this method is polynomial. □

If we restrict the bribery actions to only independent variables, the bribery problem is easy also for the cost scheme C_{LEVEL}.

Theorem 9 $(OP, IV, C_{\text{LEVEL}})$*-Bribery is in* P.

Proof If we can change only the preference orderings for the independent variables, C_{LEVEL} coincides with C_{FLIP} and thus the algorithm described in the proof of Theorem 8 , which finds the cheapest n candidates by going down in the outcome orderings of the CP-nets starting from the optimal outcome, still works. □

There are combinations of cost schemes and bribery actions for which we do not have a result for any of our one-step rules. Specifically, we have no results for C_{LEVEL} with bribery actions involving DV and C_{ANY} with any bribery actions. However, all these problems are in NP by virtue of Theorem 1 and guess and check algorithms. Given an instance of the problem and a set of bribes, we can check whether p wins in polynomial time.

Theorem 10 *(D,A,C_{LEVEL})-Bribery with $A \in \{DV, IV + DV\}$ and (D,A,C_{ANY})-Bribery with $A \in \{IV, DV, IV + DV\}$ are in* NP *when $D \in \{OP, OV, OK\}$.*

7 One-step veto and *k*-approval

We now consider the complexity of the bribery problem when we determine the winner via the veto or the k-approval voting rule. For the veto rule, we already noticed that each acyclic CP-net has exactly one optimal outcome and one worst outcome, which is the vetoed one. It is easy to find such a worst outcome. A possible method is to reverse the preference orderings all over the CP-net, obtaining a new CP-net where the optimal outcome is the worst outcome of the original CP-net. Then we determine the winner by just choosing one of the non-voted candidates (since the candidates are more than the voters).

Theorem 11 *(OV,A,C)-Bribery is in* P *when* $A \in \{IV, DV, IV + DV\}$ *and* $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{DIST}}\}$*, or if* $A \in \{IV\}$ *and* $C \in \{C_{\text{LEVEL}}\}$.

Proof The choice of IV, DV, or IV+DV, does not matter in this context. We recall that, to compute the worst candidate for each voter, it is sufficient to reverse the orderings in all his cp-statements. Let us call any CP-net on which this is performed a reversed CP-net. If the briber's preferred candidate p has no vetoes (that is, it is not optimal for any reversed CP-net), then we trivially accept (since p is, at worst, tied for a winning position). Otherwise, we consider two cases:

– If $n < 2^m$, we must bribe all vetoes for p to some other candidate. In this case there will necessarily be some candidates that receive no vetoes. Therefore, in order for p to win, he must have no vetoes.
– If $n \geq 2^m$, we consider the reversed CP-nets and we use an algorithm based on a flow network similar to the one in Theorem 8 to redistribute the vetoes. Note that the number of candidates, in this case, is polynomial in the size of the input, and thus we can use all of the candidates in the flow network. We consider one network for each $r \in \{0, \ldots, n\}$. The networks are the same as in the proof of Theorem 8 except that nodes not representing p have capacity ∞ into the sink.

□

When using One-step k-approval (OK), we determine the winner by considering the top k outcomes from each agent. Since CP-nets induce a partial order, we mean the top k according to some linearization. In the traditional bribery domain, the bribery problem for k-approval, when $k \geq 3$, is NP-complete when all the bribery costs are equal [21]. However, when agents' preferences are expressed as CP-nets, bribery schemes for k-approval can be computed in polynomial time under certain conditions.

Lemma 1 *Given an acyclic CP-net X and a constant linearization scheme across all agents and $k = 2^j$, for some $j \in \mathbb{N}$, the top k outcomes of a X are all the outcomes differing from the top one on the value of exactly j issues. Moreover, given two CP-nets in the same profile and with the same top element, they have the same top k elements.*

Proof Given a CP-net, consider any order of its issues which is compatible with its dependency graph. One can think of a CP-net as a binary string with elements labeled $x_1 > \cdots > x_i > x_{i+1} > \cdots > x_m$ regardless of the dependencies that are present (imagine the last variable in the ordering is the lowest bit of the binary string). For a given top outcome (assignment to variables $x_1, \ldots, x_m$) the next outcome will be the one with $\bar{x_m}$ and variables $x_1, \ldots, x_{m-1}$ maintaining their assignments. That is, the next outcome in terms of worsening flips will be the outcome that varies only on the last element. If $k = 2^j$, then the top k outcomes have the same values for issues $x_1, \ldots, x_{m-j}$ and differ on the values of issues $x_{m-j+1}, \ldots, x_m$. Since we can use the same topological order for any two CP-nets in a profile, if two CP-nets have the same top outcome then they have the same top k outcomes when k is a power of 2. □

A practical use of this result, apart from its use in this paper, is for the elicitation problem for CP-nets: if the ordering over the issues is known, then we only need to elicit the top outcome in order to know a voter's top k outcomes, when k is a power of 2.

Theorem 12 *When $k = 2^j$, for some $j \in \mathbb{N}$, (OK,A,C)-Bribery is in* P *if $A \in \{IV, DV, IV + DV\}$ and $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{DIST}}\}$, or if $A \in \{IV\}$ and $C \in \{C_{\text{LEVEL}}\}$.*

Proof When k is a power of two, Lemma 1 tells us that, if we fix the top outcome, the rest of the outcomes must follow in some unique order. Therefore, we treat the top k outcomes as one bundle and we apply the algorithm in the proof of Theorem 8 in order to decide the cheapest bribery scheme to elevate the bundle that includes the briber's preferred outcome p into the winning set. □

8 Conclusions and future work

We have studied the computational complexity of bribery when voters use CP-nets, an expressive and compact way to represent agents' preferences. To do this, we have generalized the traditional bribery problem to take into account several issues over which agents vote, and their inter-dependencies. We have considered five election rules, three bribery methods, and five cost schemes. For most of the combinations of these parameters, bribery in this setting is computationally easy. Our results are summarized in Table 2.

Our results are specific for each cost scheme. We plan to study possible connections between the computational complexity of reasoning with the different cost schemes.

We have also shown some interesting properties of acyclic CP-nets that can be leveraged for computational and preference elicitation reasons. In fact, when agents express their preferences as acyclic compatible CP-nets, there are cases of the bribery problem that become computationally *easier* than the single issue domains, since such CP-nets have a restricted expressive power in terms of the orderings they can induce.

There are many directions for future work. In this paper we only considered binary issues. When the issues are not binary and we follow a sequential approach, we can use a different voting rule at each level. We would like to investigate the different computational properties of combining voting rules in this sequential manner.

We also plan to study other scoring and voting rules, additional bribery actions that can also add dependencies, and the combination of weights with other voting

Table 2 Our bribery complexity results. SMw stands for SM with weighted voters, OP(A) stands for voting rule OP with bribery actions A, and similarly for OV and OK

	SM	SMw	OP(IV)
			OV(IV)
			OK*(IV)
				OP(DV,IV+DV)
				OV(DV,IV+DV)
				OK*(DV,IV+DV)

C_{EQUAL}	NP-c	NP-c	P	P
C_{FLIP}	P	NP-c	P	P
C_{LEVEL}	P	NP-c	P	$\in$ NP
C_{ANY}	P	NP-c	$\in$ NP	$\in$ NP
C_{DIST}	?	NP-c	P	P

OK* stands for OK when k is a power of 2 rules. Finally, we note that in the setting considered in this paper, the agent can only delete dependencies in response to a briber's request, while we would like to investigate the complexity of the bribery problem when agents are also allowed to add dependencies within their CP-net. We are also curious about modeling influence between agents in this domain, allowing the briber to create dependencies between agents.

Acknowledgements Portions of this work were completed while Nicholas Mattei was a graduate student at the University of Kentucky and we acknowledge their support. This material is based upon work supported by the National Science Foundation under Grant No. IIS-1107011 and CCF-1049360. The content reflects the views of the authors and not necessarily those of NSF.

This work has also been partially supported by the MIUR PRIN 20089M932N project "Innovative and multi-disciplinary approaches for constraint and preference reasoning" an by the project "Intelligent preference reasoning for multiagent decision making" funded by the University of Padova.

NICTA is funded by the Australian Government through the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
2. Arrow, K.J.: Social Choice and Individual Values. Wiley (1963)
3. Arrow, K.J., Sen, A.K., Suzumura, K.: Handbook of Social Choice and Welfare. Elsevier, North-Holland (2002)
4. Bartholdi, J., Tovey, C., Trick, M.: The computational difficulty of manipulating an election. Soc. Choice Welf. **6**(3), 227–241 (1989)
5. Birrell, E., Pass, R.: Approximately strategy-proof voting. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 67–72 (2011)
6. Boutilier, C., Bacchus, F., Brafman, R.: UCP-networks: a directed graphical representation of conditional utilities. In: Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI 2001), pp. 56–64 (2001)
7. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. **21**(1), 135–191 (2004)
8. Brafman, R.I., Rossi, F., Salvagnin, D., Venable, K.B., Walsh, T.: Finding the next solution in constraint- and preference-based knowledge representation formalisms. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010) (2010)
9. Brams, S., Kilgour, D., Zwicker, W.: The paradox of multiple elections. Soc. Choice Welf. **15**(2), 211–236 (1998)
10. Brandt, F., Conitzer, V., Endriss, U.: Computational social choice. In: Weiss, G. (ed.) Multiagent Systems. MIT Press (2012)
11. Christian, R., Fellows, M., Rosamond, F., Slinko, A.: On complexity of lobbying in multiple referenda. Rev. Econ. Des. **11**(3), 217–224 (2007)
12. Conitzer, V., Lang, J., Xia, L.: How hard is it to control sequential elections via the agenda? In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 103–108 (2009)
13. Conitzer, V., Sandholm, T., Lang, J.: When are elections with few candidates hard to manipulate? JACM **54**(3), 1–33 (2007)
14. Conitzer, V., Xia, L.: Approximating common voting rules by sequential voting in multi-issue domains. In: Proc. International Symposium on Artificial Intelligence and Mathematics (ISAIM 2012)—Special Session on Computational Social Choice (2012)
15. Dorn, B., Schlotter, I.: Multivariate complexity analysis of swap bribery. Algorithmica **64**, 126–151 (2012)

16. Downey, R., Fellows, M.: Parameterized Complexity. Springer (1999)
17. Elkind, E., Faliszewski, P.: Approximation algorithms for campaign management. In: Proceedings of the 6th International Workshop on Internet and Network Economics (WINE 2010), pp. 473–482 (2010)
18. Elkind, E., Faliszewski, P., Slinko, A.: Swap bribery. Algorithmic Game Theory, pp. 299–310 (2009)
19. Erdélyi, G., Fernau, H., Goldsmith, J., Mattei, N., Raible, D., Rothe, J.: The complexity of probabilistic lobbying. In: Proc. 1st International Conference on Algorithmic Decision Theory (ADT 2009), pp. 86–97 (2009)
20. Faliszewski, P.: Nonuniform bribery. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 1569–1572 (2008)
21. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.: How hard is bribery in elections? J. Artif. Intell. Res. **35**, 485–532 (2009)
22. Faliszewski, P., Hemaspaandra, E., Schnoor, H.: Copeland voting: ties matter. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008) (2008)
23. Gibbard, A.: Manipulation of voting schemes: a general result. Econometrica **41**(4), 587–601 (1973)
24. Lang, J.: Vote and aggregation in combinatorial domains with structured preferences. In: Proceedings of the 20nd International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 1366–1371 (2007)
25. Lang, J., Pini, M.S., Rossi, F., Salvagnin, D., Venable, K.B., Walsh, T.: Winner determination in voting trees with incomplete preferences and weighted votes. JAAMAS **25**(1), 130–157 (2012)
26. Lang, J., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Winner determination in sequential majority voting. In: Proceedings of the 20nd International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 1372–1377 (2007)
27. Lang, J., Xia, L.: Sequential composition of voting rules in multi-issue domains. Math. Soc. Sci. **57**(3), 304–324 (2009)
28. Magrino, T., Rivest, R., Shen, E., Wagner, D.: Computing the margin of victory in IRV elections. In: Proc. Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (2011)
29. Mattei, N., Pini, M.S., Rossi, F., Venable, K.B.: Bribery in voting over combinatorial domains is easy. In: Proceedings of the 11th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012), pp. 1407–1408 (2012) (Extended Abstract)
30. May, K.: A set of independent necessary and sufficient conditions for simple majority decisions. Econometrica **20**(4), 680–684 (1952)
31. Obraztsova, S., Elkind, E.: On the complexity of voting manipulation under randomized tie-breaking. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 319–324 (2011)
32. Obraztsova, S., Elkind, E., Hazon, N.: Ties matter: complexity of voting manipulation revisited. In: Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), pp. 71–78 (2011)
33. Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Incompleteness and incomparability in preference aggregation. In: Proceedings of the 20nd International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 1464–1469 (2007)
34. Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Incompleteness and incomparability in preference aggregation: complexity results. Artif. Intell. **175**(7–8), 1272–1289 (2011)
35. Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Possible and necessary winners in voting trees: majority graphs vs. profiles. In: Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), pp. 311–318 (2011)
36. Satterthwaite, M.: Strategy-proofness and arrow's conditions: existence and correspondence theorems for voting procedures and social welfare functions. J. Econ. Theory **10**(2), 187–216 (1975)
37. Xia, L.: Computing the margin of victory for various voting rules. In: Proceedings of the 13th ACM Conference on Electronic Commerce (EC 2012), pp. 982–999 (2012)
38. Xia, L., Conitzer, V.: Strategy-proof voting rules over multi-issue domains with restricted preferences. In: Proceedings of the 6th International Workshop on Internet and Network Economics (WINE 2010), pp. 402–414 (2010)
39. Xia, L., Conitzer, V.: Determining possible and necessary winners given partial orders. J. Artif. Intell. Res. **41**, 25–67 (2011)