Chapter 4 Bribery and Manipulation in Combinatorial Domains

This chapter details work on 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 their domains, and agents' preferences are represented as CP-nets. This work includes model building; extending and defining aggregation methods and bribery methods for these models; and complexity results of reasoning in these domains. We find that this domain structure, which may lead to an exponential number of candidates in the size of the input, causes many existing computational results for bribery to break down. We provide new algorithms and complexity results which show that, in most cases, bribery in combinatorial domains is easy. A summary of our results is shown in Table 4.1 and Table 4.2. This also holds for some cases of *k*-approval, where bribery is difficult in traditional domains. Portions of this work have been previously published as both an extended abstract [88] and as an invited special session paper [89].

4.1 Voting in Combinatorial Domains

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, and wine should be served for a group meal [18, 82]. 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 agents to compactly represent their preferences over many issues that may have conditional dependencies [15, 90].

As we have seen, bribery and manipulation directly compromise the security and robustness of an election system. These questions are ways that agents (outside or inside an election) can affect the election's outcome [7,52]. 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 p wins the election. This problem was introduced to computational social choice by Faliszewski et al. [52] and has become a metric by which the security of election rules is judged [39, 51, 85]: 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.

Manipulation occurs when one or more voters attempt to vote strategically in order to affect the result of the election [7,33]. Recall the **Constructive Coalitional Manipulation (CCM)** problem [33]: given a set of agents X with fixed votes, a set Y of agents with unfixed votes, and a candidate p, is there is an assignment of votes to the agents of Y such that running the election on the complete set of votes (profiles) $X \cup Y$ 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 do so for a unit cost, those that cannot modify their vote have a bribery price of ∞, and the budget is equal to the size of the manipulator set. While every reasonable voting rule is manipulable [3], a rule may be said to be resistant to manipulation if it is computationally difficult to decide how to manipulate [7]. The complexity of manipulation was first studied by Bartholdi et al. [7] with further study by Conitzer et al. [33] and is another metric for election security. These problems have been studied extensively in computational social choice, however, the literature is sparse when elections have combinatorially structured domains.

The setting we study in this chapter is novel for several reasons. We allow sets of candidates which are the Cartesian product of the domains of a set of possible decisions. This raises several interesting questions as to what it means to compute a winner and what bribery means in these combinatorial domains. The use of structured, combinatorial preferences leads to instances where some voters, depending on their preference dependencies, cannot be bribed to vote for any arbitrary candidate. This restriction, along with the potentially exponential number of candidates, breaks some of the existing algorithms for bribery and manipulation.

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 overarching winner determination strategies.

Sequential Methods: A sequential method aggregates each agents' preference issue-by-issue. This is akin to how amendments to a bill are passed in the US Congress [18]. The final decision is the sum total of all the sequential decisions. In this chapter we consider a 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 [82]. 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, CP-nets do not necessarily lend themselves 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 [43, 52]. 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 four different bribery cost schemes.

$C_{\mathbf{EQUAL}}$: Any amount of change in a CP-net is a unit cost.

$C_{\mathbf{FLIP}}$: 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.

In the rest of this section we provide an overview of structured preference models, specifically CP-nets, for compactly expressing preference in combinatorial domains and their use in voting procedures. In Section 4.2 we precisely define our domain and formulate our three bribery actions and four cost schemes that are used in combinatorial bribery. Section 4.3 gives the bribery and manipulation problems within our model. Section 4.4 gives our results about the complexity of computing optimal bribery and manipulation schemes in our models. Our results section is broken up according to voting rule and aggregation method in order to provide a cohesive narrative. Section 4.4.4 expands our model to encompass domains where variables are non-binary and provides results about the composition of multiple voting rules. We end in Section 4.5 which details the changes in complexity from traditional bribery domains to those when agents' preferences are represented as CP-nets.

4.1.1 Structured Preferences

Combinatorially structured election domains are used in many places including the United States House of Representatives [18]. This election structure considers several possibly independent variables of some whole described by the assignment of each of the variables to some value. The canonical example in the ComSoc community is choosing the appetizer, main course, desert, and wine for a meal. Each course has several alternatives and the combination of all possible choices for all possible courses creates the set of alternatives (or candidates). This sequential framework creates a possibly exponential number of complete configurations of meals in the description of the problem instance.

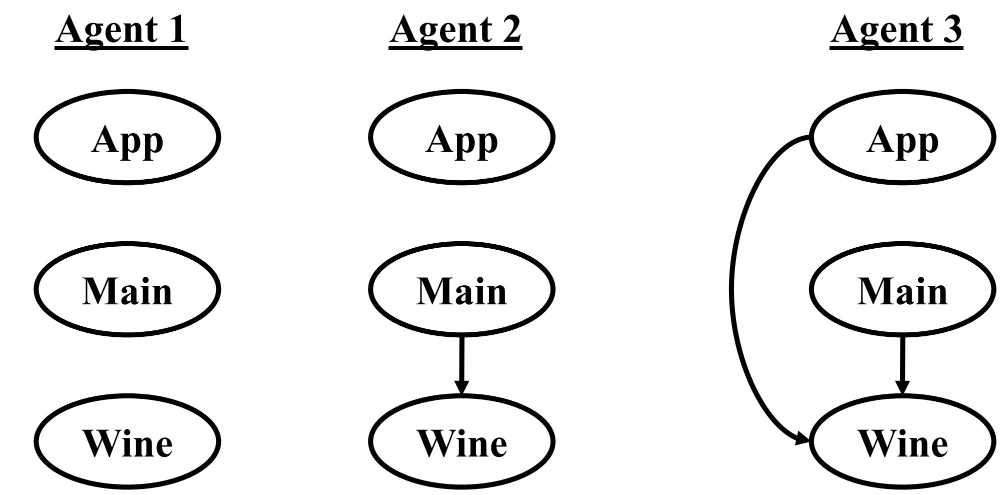
Sequential elections were first introduced and studied in computational social choice by Lang [79]. Lang provided a comprehensive axiomatic description of several voting rules when extended to combinatorial domains when agents' preferences are expressed as CP-nets [79]. In later work, Lang and Xia considered the properties of several election rules and how these properties change when rules are used as part of a sequential election domain [82]. Additional work by Xia et al. investigates the properties of sequential election rules when cyclic preferences are allowed [137] and axiomatic characterizations of strategy-proof sequential voting rules [136].

While bribery has not been specifically investigated in any domain with sequential elections, Conitzer et al. investigated the control problem in sequential voting domains [29] when agents' preferences are recorded with a network structure similar to CP-nets. While Xia et al. have investigated the possibility of strategic voting by a single voter in sequential voting when agents' preferences are recorded as CB-nets [138]. Dalla Pozza et al. investigate sequential voting domains when agents' preferences are represented as soft constraints [37].

In our model of elections under uncertainty in Section 3.1, observe that a complete agenda with multiple referenda is similar to the combinatorial domains we study in this chapter. However, in the uncertain election model, all issues are independent whereas in the combinatorial domains in this chapter allow for issues to have conditional dependencies upon one another. In some cases one model can be described with the other (deterministic instances of the uncertain information model are equivalent to the combinatorial model with all independent variables). This subcase, however, does not take advantage of the fidelity of either model. We think of them as models designed to investigate different types and amounts of information that may be available to an outside agent.

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 [15], 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).

Figure 4.1: An example of a CP-net with three agents expressing *O*-legal profiles over three binary variables.



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

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.

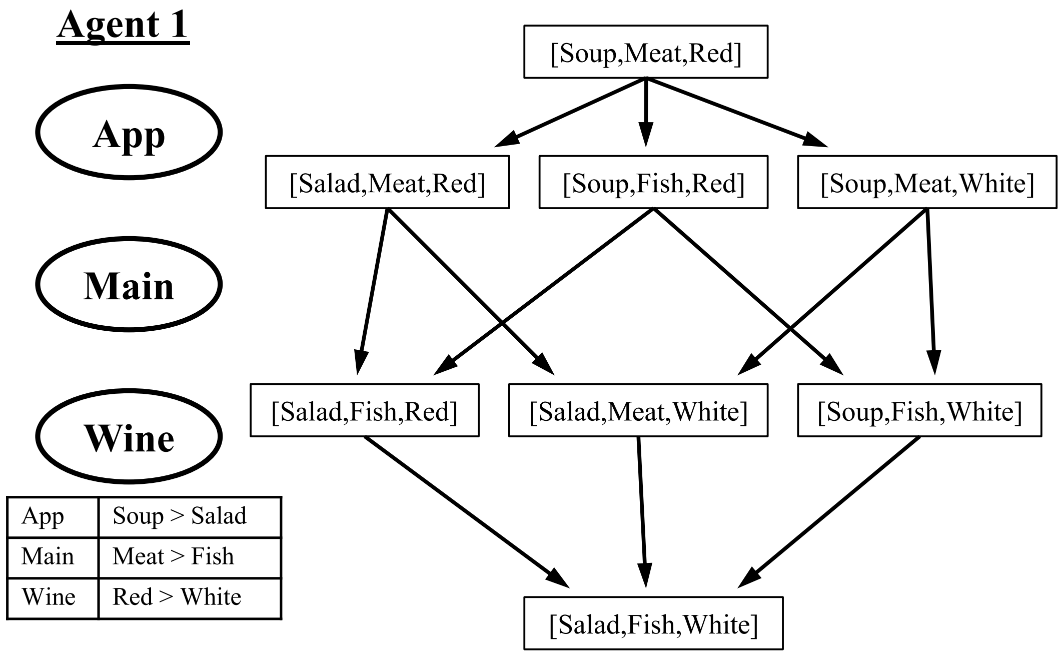
Formally, a CP-net has a set of issues or variables $m = \{x_1, \ldots, x_n\}$, and each issue has a finite domain $D(x_1), \ldots, D(x_n)$. Each issue x_i has a set of parent issues $Pa(x_i)$ whose assignment can affect the ordering over the values of x_i. This set of dependencies defines a graph where each x_i has $Pa(x_i)$ as its immediate predecessors. In general, CP-nets allow for cyclic dependencies within the implied graph. However, in this work we only allow acyclic dependency graphs. Given the implied graph structure an agent specifies a total order (strict linear order) over the values of each x_i for each complete assignment of the variables in $Pa(x_i)$. These orders are referred to as cp-statements. A CP-net is **compact** if the maximum number of parents of a feature is bounded from above by a constant. This formalism allows for a superpolynomial number of outcome assignments in the description size of the CP-net. If we allowed for an unbounded number of parents for each variable, since we have to enumerate all linear orders for all combinations of parent variables, our tables would be exponential in size.

Figure 4.1.1 shows an example of a CP-net and corresponding cp-statements. In this graph we have three issues $m = \{App, Main, Wine\}$. The domains for the variables are $D(App) = \{Soup, Salad\}$, $D(Main) = \{Fish, Meat\}$, and $D(Wine) = \{Red, White\}$. We have three different agents and each agent has its own set of dependency relations within the preference graph. Agent 1 has no preferential dependencies and therefore none of his variables have parent variables ($Pa(App) = \emptyset$ for Agent 1). Agent 2's preferences on wine are dependent on his assignment for main course. The encoding of Agent 2's preferences make the statement that, "I prefer red wine to white if meat is served." So 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 graph. Recall that, for each variable, a complete assignment must be provided for all assignments of the parent variables. This is why Agent 3's preference table is 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 $Main$.

In general we consider issues with only binary domains in this chapter. 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 preference over an outcome space of cardinality exactly 2^m, where m is the number of issues.

In general, finding the optimal (most preferred) outcome of a CP-net is NP-hard [15].

Figure 4.2: An example of a CP-net with the corresponding graph representing the partial order between all possible outcomes.

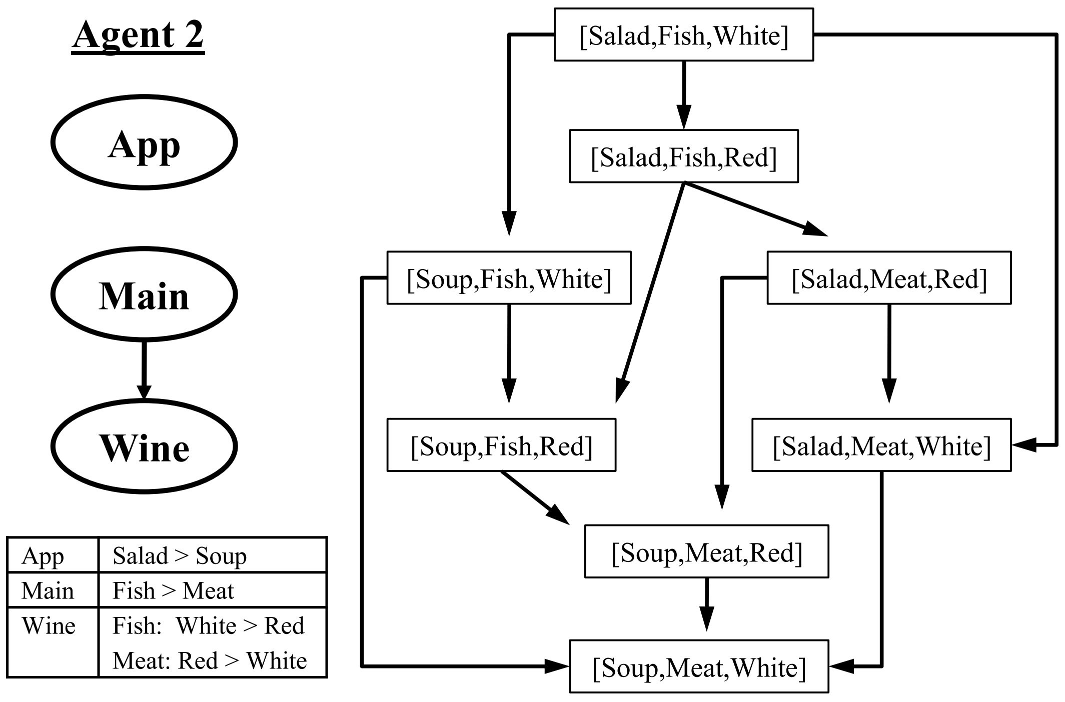


App	Soup > Salad
Main	Meat > Fish
Wine	Red > White

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 Figure 4.1.1. 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. Figure 4.1.1 illustrates the partial order over all the outcomes defined by the CP-net of Agent 1. The top outcome according

Figure 4.3: An example of a CP-net with the corresponding graph representing the partial order between all possible outcomes. Ties are broken with independent variables being considered more important than dependent variables.



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

to the CP-net is $\{Soup, Meat, Red\}$. The arrows from this outcome represents all the possible worsening flips from the top outcome. 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 (occupy the same position in the partial order).

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 [17]. In a CP-net, all outcomes may not be ordered since the CP-net only implies a partial order and so some assignments may be incomparable. In some cases, we may require a total order over the outcome ordering and in these cases we enforce some kind of linearization as is standard [20, 34]. If finding the next best outcome according to the linearization scheme is computationally easy then it is also easy to find the top k outcomes when k is polynomially bounded.

A standard way to define a linearization is to provide a total order over the variables and then to linearize incomparable elements using a lexicographical approach based on this total order and on the conditional preferences. We illustrate this in Figure 4.1.1 where we show Agent 2's CP-net and the induced graph according to worsening flips. The graph also shows the partial order when we consider independent variables more important than dependent variables. We have broken the tie between $\{Salad, Fish, Red\}$ and $\{Soup, Fish, White\}$ using this rule. Each of these assignments contains one variable with a less preferred assignment. The reason that $\{Salad, Fish, Red\}$ precedes $\{Soup, Fish, White\}$ in the partial order is because $\{Salad, Fish, Red\}$ has a less preferred assignment in a dependent variable (less important). If we were to linearize the order in Figure 4.1.1 into a strict linear order using a lexicographic linearization method by variable name we get: $\{Salad, Fish, White\} > \{Salad, Fish, Red\} > \{Salad, Meat, Red\} > \{Soup, Fish, White\} > \{Salad, Meat, White\} > \{Soup, Fish, Red\} > \{Soup, Meat, Red\} > \{Soup, Meat, White\}$.

Given a set of issues, domains, and some preferred outcome p, there is always a CP-net that has p as its optimal outcome [15]. 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 that 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 qualitative nature, especially in the context of preference elicitation.

4.1.2 Winner Determination and Voting with CP-nets

Voting theory provides a myriad voting rules to aggregate agents' preferences. We survey some of these methods in Section 2.2.1. Recall that 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 [91]. 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 chapter we consider the following voting rules, modified slightly to work when agents' express their preferences as CP-nets. Each candidate when voting in combinatorial domains is a complete assignment to all variables within the domain.

Plurality: The candidate ranked first receives a point. The candidate(s) with the most points win 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(s) 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(s) with the most points win the election.

The study of voting theory, as discussed in Section 2.2.1, 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 chapter we focus on combinatorial voting domains and their resistance to bribery.

The model we consider in this chapter has a n agents with preferences over a common set of candidates. The candidate set has a combinatorial structure: there is a common set of

m binary issues and the set of candidates is the Cartesian product of the variable domains. Each candidate (or outcome) is an assignment of values to all issues, thus we have 2^m candidates.

Each agent expresses its preferences over the candidates via a compact acyclic CP-net. Moreover, when we use a sequential approach to voting we require additional restrictions on the CP-nets. The CP-nets for sequential methods must be compatible with one another: there is a total ordering O over the issues such that, in each CP-net, each issue must be independent of all issues following it in the ordering O and no dependency graph may contain cycles.

A **profile** (P,O) is a collection $P_1,\dots,P_n$ 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 and was introduced by Lang [79]. Notice that the CP-nets appearing in such profiles do not necessarily have the same dependency graphs. This is illustrated in Figure 4.1.1, our running example. Each of the three agents has the same ordering over the issues $O = \{App, Main, Wine\}$ but each agent has a different dependency graph. Moreover, each of the agents' CP-nets combine to form an O-legal profile: the CP-nets are acyclic, follow an ordering, and are compact.

It is worthwhile to clarify our notion of an agent: an agent is only a CP-net handler. He has the ability to specify a CP-net over a given set of binary issues and to compute the optimal outcome, the top k outcomes, and the worst outcome of his CP-net. He does not handle the explicit partial order over all the outcomes of the CP-net, but just the CP-net, which is a compact representation of the ordering.

To determine the winning outcome, we consider two different approaches: sequential and one-step rules. Within these, we define four distinct voting rules when selecting a winning outcome. Notice that the O-legality condition is not required for the one-step approaches; in these settings, the CP-nets in a profile can have very different dependency structures. Tie-breaking, as discussed in Section 2.2.1 is a complex issue and we discuss it for our domain when we formally define our problem in Section 4.3.

Sequential Majority (SM)

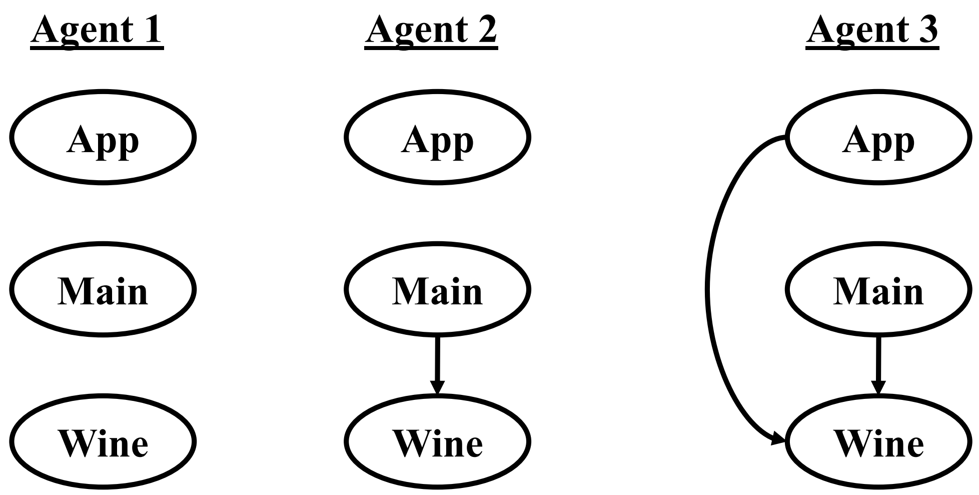
Sequential Majority describes Lang's sequential procedure [79] instantiated on binary issues. For SM we consider only O-legal profiles. For each issue in the ordering O, we select a winning assignment using majority (since issues are binary). The value chosen for an issue is returned to all agents who then fix the issue to that value in their respective CP-nets. When voting has completed on all issues, the winning outcome is defined by the issue instantiations chosen by majority at each of the voting steps.

Example 4.1.1 *Consider our running example of three agents illustrated in Figure 4.1.1. 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 and $Salad$ is preferred by a vote of two to one. 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 $\{Salad, Fish\}$. This has caused two of Agent 1's preferences to be reversed but we do not need to update his CP-net as it contains no dependencies. For Wine we see that Agent 2's preferences are $White > Red$ given our assignment to $Fish$. Agent 3's preference is for $White$ as well and now we have a majority for $White$ of two to one. The winning outcome for this example with the SM procedure is $\{Salad, Fish, White\}$.*

One-step Plurality (OP)

In OP 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, merely that agents be able to compute their top outcome.

Figure 4.4: An example of a “reversed” CP-net with three agents expressing *O*-legal profiles over three binary variables with all cp-statements reversed.



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

Example 4.1.2 *Consider our running example of three agents illustrated in Figure 4.1.1. By investigation we see that Agent 1 will vote* $\{Soup, Meat, Red\}$*, Agent 2 will vote* $\{$*Salad, Fish, White*$\}$*, and Agent 3 will vote* $\{Salad, Fish, White\}$*. Collecting these votes under the plurality rule the winning outcome with two votes is* $\{Salad, Fish, White\}$*.*

One-step Veto (OV)

In OV 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 Figure 4.1.2. 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. However, since we are using this as a social choice procedure, we only need to select one alternative from the winning set, which we can do in polynomial time.

Example 4.1.3 *Consider our running example of three agents illustrated in Figure 4.1.1. First each agent must reverse all the cp-statements in their CP-nets. This will leave the agents with the CP-nets illustrated in Figure 4.1.2. We see that Agent 1 will veto* $\{Salad, Fish, White\}$ *while Agent 2 and Agent 3 will both veto {Soup, Meat, White}. In this case we have veto's 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 employ a lexicographic tie breaking rule.*

One-step k-approval (OK)

We ask each agent to provide their top k outcomes according to their CP-net, and a linearization of the outcome ordering (unless otherwise noted we always assume that our linearization is according to the variable ordering, then lexicographically). We require the linearization here because it is possible that k is not a round power of two. It is also possible, though we do not discuss it in this work, for each agent to individually employ their own linearization procedure. In this case we could have that k includes some, but not all, elements of an incomparable set in the partial order.

Example 4.1.4 *Consider the situation where* $k = 2$ *in our running example illustrated in Figure 4.1.1. We see that Agent 1 will approve of* $\{Soup, Meat, Red\}$ *and {Soup, Meat, White}. Agent 2 and Agent 3 will approve of* $\{Salad, Fish, White\}$ *and* $\{Salad, Fish, Red\}$*. Therefore, the winning set consists of the two options chosen by both Agent 2 and 3.*

4.2 Bribery and Manipulation

The bribery problem we consider in this chapter has three parameters: the way a winner is chosen from the given profile, the bribery actions, and the cost scheme for the bribers' requests. In this section we extend the traditional notions of bribery in order to account for combinatorial domains and our structured preference representation. Bribery is usually considered in domains with only one issue to be decided [52] or, as in Section 3.1, multiple independent issues. It is a non-trivial exercise to define bribery in the domain considered in this chapter.

4.2.1 Bribery Actions

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. We define the bribery actions as changes in the total order of some cp-statements. If the briber could ask for any change in the ordering induced by the CP-net, some of these changes could be computationally hard or impossible to accomplish via changes to the CP-net. Therefore, we define the bribery actions as changes made directly to the cp-statements within the CP-net of an agent. Since a cp-statement gives a total order for the domain of an issue, the briber asks for a new total order over that domain. In this section 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 affect larger changes in the final preference order and, since they are independent, agents have "pure preferences" over them, independent of 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 assignment 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 bribe is available then there are outcomes that may not be able to be elevated to the most preferred outcome, as illustrated in the following example.

Example 4.2.1 *Consider our running example from Figure 4.1.1. Suppose $p = \{Soup, Meat, White\}$ and we need to bribe some of the agents to have this as their top outcome using the SM rule.*

When considering only IV bribes, all of Agent 1's preferences can be changed since they 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 Agent 2's most preferred assignment is $Wine = Red$, given $Main = Meat$. Since we are only allowing IV bribes we cannot make Agent 2's preferences match p. For Agent 3 we have the same problem as Agent 2. Though we can set Agent 3's assignment to $App = Soup$ and $Main = Meat$, this causes $Wine = Red$ in the CP-net. We cannot make p the winner of the election.

Consider the case of only DV bribes. In this case we cannot change any of Agent 1's preferences because all his preferences are independent. Likewise, we cannot bribe Agent 2 or Agent 3's assignment of App because no agent has App as a dependent variable. In this case we could change the assignment of $Wine$ but because we cannot change the assignment of any other variable, we cannot make p a winner.

In the case of IV+DV bribes we can make p a winner. We make the same bribes to the agents as we did in the IV example. This time, 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 a winner in the election.

4.2.2 Cost Schemes

In traditional bribery domains, the cost of the bribery actions is either fixed for any amount of change for each agent [52], variable per agent for any amount of change [51], or it depends on the number of swaps to be performed on the current total order [43]. 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. In fact, a formalism closer to the one defined in Chapter 3, where agents have cost associated to the change in probability of a vote, is not sufficient in the CP-net context either. Therefore, in this chapter we consider four 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 four cost schemes we investigate in this chapter are listed here from least expressive to most expressive.

$C_{\textbf{EQUAL}}$**:** 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 [52].

$C_{\textbf{FLIP}}$**:** 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. [43] applied to CP-nets instead of strict linear orders.

$C_{\textbf{LEVEL}}$: We expect an issue that is closer to being independent is more important, as in some other structured preference formalisms [14], than a issue buried deep 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 a issue recursively as:

$$level(x) = \begin{cases} 1 & \text{if } x \text{ is an independent issue;} \\ i+1 & \text{if all parents of } x \text{ are in levels } \{1,\ldots,i\} \\ & \text{and there is a parent in level } i. \end{cases} \tag{4.1}$$

We can then precisely define the cost of flipping an issue as $\sum_x flip(x) \times (k+1-level(x))$, 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.

$C_{\textbf{ANY}}$: 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 $v_i \rightarrow \mathbb{Z}_0^+$ so that modifying any variable in the instance has its own associated price. This is similar to the nonuniform bribery model introduced by Faliszewski [51]

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 $\vec{Q} \in \mathbb{Z}^+$ of size 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. [52].

In many variations of the bribery and manipulation problems it is standard to attach weights to each voter [33, 52]. Weights usually represent a group of voters who all think alike, or voters with differing levels of importance such as shareholder votes for publicly traded companies. We will sometimes assume a weight vector is provided $\vec{W} \in \mathbb{Z}^+$ where $W[i]$ represents the weight associated to each voter i. When we use weighted voters we require that p be elected according to the given voting rule.

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. Consider the CP-net and associated ordering presented for Agent 2 in Figure 4.1.1. In the outcome ordering the optimal ordering is $\{Salad, Fish, White\}$ and $\{Salad, Meat, Red\}$ dominates $\{Salad, Meat, White\}$ in the CP-net and therefore $\{Salad, Meat, White\}$ is farther away from the optimal outcome in the partial order. To make $\{Salad, Meat, White\}$ the optimal outcome we only need to flip one cp-statement ($Fish \rightarrow Meat$). However, to make $\{Salad, Meat, Red\}$ the optimal outcome in the CP-net, we require 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 $2 \times 1 + 1 = 3$ (one independent and one dependent flip). Therefore, the cost schemes C_{LEVEL} and C_{FLIP} (and the more general C_{ANY}) do not respect the distance-by-flips notion.

4.3 The Combinatorial Bribery Problem

We are now ready to formally state the bribery problem we study in this chapter. 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 $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}\}$.

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 $B \in \mathbb{Z}^+$, a preferred outcome p, and (when necessary) a bribing cost vector $\vec{Q} \in \mathbb{Z}^+$, and weights $\vec{W} \in \mathbb{Z}^+$.

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 according to $A \in \{IV, DV, IV+DV\}$, paying according to the cost scheme $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}\}$ and bribing cost vector $\vec{Q}$, weight $\geq W$ and without exceeding budget B?

The CCM problem is defined analogously for our domain with the condition that all entrants in the manipulator set have a unit cost to bribe and all other voters have infinite cost. This means that the CCM problem is actually a sub-problem of the bribery problem in this case.

Name: (D)-Manipulation

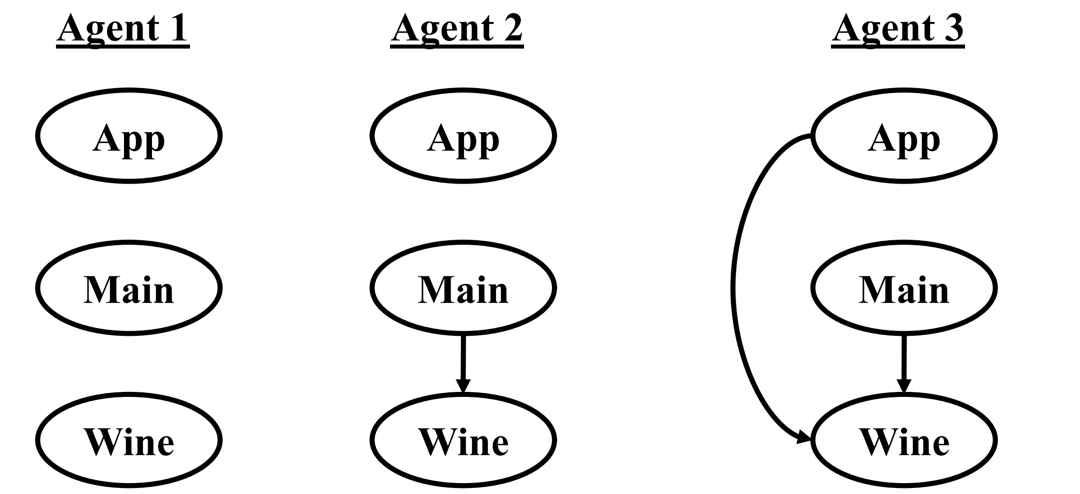
Given: A preferred outcome p and profile (P,O), where P is divided into two disjoint sets X and Y where $||X|| + ||Y|| = n$. The voters in X have preferences expressed as compact, acyclic CP-nets with m binary issues that follow the ordering O over the issues (when necessary) and the voters in Y have no preferences.

Question: Are there CP-nets for the agents in Y that are consistent with the ordering O (when necessary) such that p is made a winner in profile (P,O) with winner determination rule $D \in SM, OP, OK, OV$?

We assume, as in other works on bribery such as those by Elkind et al. [43] and Faliszewski et al. [52], 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 general case. While we obtain mostly easiness results under this winner model (as did Bartholdi et al. [7]), a complete investigation of tie-breaking and its effect on the complexity of manipulation is outside the scope of this chapter. In fact, the careful study of tie-breaking mechanisms is extremely complex as shown by Obraztsova et al. [98] among others [58] and requires its own course of study.

Example 4.3.1 *Recall our running example with the CP-nets depicted below.*

Figure 4.5: CP-net with three agents expressing O-legal profiles over three binary variables for Example 4.3.1.



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

Given these profiles and $O = \{App, Main, Wine\}$ we can consider several examples of our bribery problem parameterized by different attributes.

(SM-IV-C_{EQUAL})-Bribery: *Consider a case where $B = 10$, $\vec{Q} = [5,5,5]$, 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. By investigation we see that Agent 1 has exactly p as its top outcome so we do not need to bribe him at all. Additionally, we see that if we give bribes to either Agent 2 or Agent 3 on the App and Main, they will change Wine $\rightarrow$ Red. Since we are using C_{EQUAL} and $Q[2] = Q[3] = 5$ we can make both assignment changes in either agents' cp-statements 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 a case where $B = 5$, $\vec{Q} = [1,1,2]$, 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 a case where $B = 3$, $\vec{Q} = [2,1,2]$, and $p = \{Salad, Meat, White\}$. Currently, no agent has this as their top preference and p is not winning. Notice that, in order to set Agent 1 to have p as his top preference requires two bribes to independent variables. Since all variables for Agent 1 are at level one out of the two levels of the CP-nets, the cost of this bribe is $Q[1] \times 2 = 4$. However, bribing either Agent 2 or Agent 3 requires a bribe at level one and a bribe at level two and therefore the cost of the bribe for Agent 2 is $Q[2] \times (1 \times 1 + 1 \times 2) = 3$ while Agent 3's bribing cost is $Q[3] \times (1 \times 1 + 1 \times 2) = 6$. We chose to bribe Agent 2 to now 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 less than our budget, so this is a "yes" instance.*

4.4 Results

In the rest of the chapter we study the computational complexity of this problem, considering all possible combinations of winner determination rules in $D \in \{SM, OP, OV, OK\}$, bribery actions in $A \in \{IV, DV, IV + DV\}$, and cost schemes in $C \in \{C_{\mathrm{EQUAL}}, C_{\mathrm{FLIP}}, C_{\mathrm{LEVEL}}, C_{\mathrm{ANY}}\}$. Our results are summarized in Table 4.1 and Table 4.2. Note that in some cases we are still missing results and we will address these gaps while we unfold our results.

We begin by looking at the complexity of winner determination in our domain. We then consider the complexity of finding optimal bribery actions in our domain. After these preliminaries, we consider each voting rule in turn and conclude our results section with some extensions to non-binary domains.

4.4.1 Winner Determination and Changing a Vote

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 4.4.1 *Winner determination is in P for SM, OP, OV, and OK (when k is bounded from above by a constant).*

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

For OP, we need to compute the optimal outcome for each acyclic CP-net, which is polynomial. Then plurality is applied, in time polynomial in the number of voters.

The winner according to One-step Veto (OV) can be obtained by reversing each total order in all cp-statements of the CP-nets. Thus, 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 time polynomial in the size of the CP-nets. 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. In some cases enumerating the entire winning set could take exponential time. However, since we assume p wins all ties we can determine if p is in the winning set by looking at the polynomial number of candidates who receive vetoes. If p is not in the winner set, then we randomly choose some other outcome receiving no vetoes as the winner of the election. This again takes polynomial time with respect to the size of the input.

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: each agent can just start from the optimal outcome and find the best k outcomes by applying the *next* operation $k-1$ times [17]. Therefore, we can produce the top k lists and apply the voting rule in polynomial time. ❑

In a bribery scenario, the briber has a desired candidate p. To make p the overall winner, the briber may need to ask some agents to change their current vote to a vote for p or to vote for some other candidate, paying for this change according to the cost scheme. In non-combinatorial bribery scenarios, agents specify their preferences explicitly and, therefore, the cost of bribery is easy to compute. In the four cost schemes that we have proposed, however, this computation is not so straightforward.

Theorem 4.4.2 *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 $C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}$, or C_{ANY}.*

Proof. Assume we can use all bribery actions, $IV + DV$. Therefore, any CP-net can be changed to vote for p (that is, to make p its optimal outcome).

For C_{EQUAL}, if p is already the top outcome of the CP-net, the cost is 0, otherwise it is 1. To find the p sweep through the CP-net and assign each variable to have the same assignment as in p.

For C_{FLIP}, we need to compute the minimum number of flips to be performed on the cp-statements within the CP-net in order to make p the top outcome. 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, after all its parents are processed, and do the same in its relevant cp-statement. At the end, p is the top outcome and the number of flips performed is the minimum cost to make p win.

Table 4.1: Bribery complexity results for Sequential Majority and Weighted Sequential Majority.

	Sequential Maj.	Theorem	Weighted Sequential Maj.	Theorem
C_{EQUAL}	NP-complete	4.4.4	NP-complete	4.4.5 and 4.4.6
C_{FLIP}	P	4.4.3	NP-complete	4.4.5 and 4.4.6
C_{LEVEL}	P	4.4.3	NP-complete	4.4.5 and 4.4.6
C_{ANY}	P	4.4.3	NP-complete	4.4.5 and 4.4.6

For C_{LEVEL} or C_{ANY}, finding the flips is 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.

If we only had access to IV or DV, as soon as a necessary flip cannot be performed, we know that p cannot be the optimal outcome. ❑

4.4.2 Sequential Rules

In this section we consider the computational complexity of bribery when we determine the winner via sequential majority. A summary of our results can be found in Table 4.1.

Theorem 4.4.3 *(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 $\vec{Q}$), 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). This allows us to effectively separate the levels so we can consider bribery for each variable independently of all other variables.

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 agents' 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 agents' 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 agents' bribery prices if p cannot be made its best outcome given the restrictions. ❑

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 agents' CP-nets to change for a particular issue is strictly related to this same decision for another issue: if we change the same agents' CP-net for multiple issues, 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 since we are trying to minimize the total number of agents whose CP-nets we need to modify. This makes the bribery problem computationally hard.

Recall the OPTIMAL LOBBYING (OL) problem introduced by Christian et al. [24].

Name: OPTIMAL LOBBYING (OL)

Given : An $n \times m$ $0/1$ matrix E and a $0/1$ vector $\vec{x}$ 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 $\vec{x}$ is the target group decision.

Parameter: $k \in \mathbb{Z}^+$: the number of agents to be influenced.

Question: Is there 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 $\vec{x}$?

This problem is shown to be $W[2]$-complete via a reduction from k-DOMINATING SET [24]. We give a polynomial reduction from OL to our bribery problem, thus showing that our bribery problem is NP-complete [40]. In fact, since OL is $W[2]$-complete for parameter k and our reduction is parameter preserving, all the bribery problems in the following proof are $W[2]$-hard with parameter B.

Theorem 4.4.4 *$(SM, IV, C_{\text{EQUAL}})$-Bribery, $(SM, DV, C_{\text{EQUAL}})$-Bribery, and $(SM, IV + DV, C_{\text{EQUAL}})$-Bribery are NP-complete.*

Proof. Membership in NP is an immediate consequence of Theorem 4.4.1 and a guess and check algorithm.

To show completeness, we provide a polynomial reduction from OL. We start with $(SM, IV, C_{\text{EQUAL}})$-Bribery. Given an instance $(E, \vec{x}, k)$ of OL, we construct an instance of

$(SM, IV, C_{\text{EQUAL}})$-Bribery containing CP-nets with only independent variables. The set 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 $p = \vec{x}$.

Thus, p wins the election if and only if there is a selection of k rows of E such that $\vec{x}$ becomes the winning agenda of the OL instance. Therefore $(SM, IV, C_{\text{EQUAL}})$-Bribery is NP-complete.

The same reduction works for IV+DV as well. For DV, we choose an instance of the bribery problem as above, except we add one more issue and each voter therefore has an additional independent variable on which all others 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 opposite preference. The last difference is that now p is $1\vec{x}$. 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 bribery with all cost schemes is computationally difficult. We denote by **weighted-X** the bribery problem X with weighted agents.

Faliszewski et al. [52] show NP-completeness of the plurality-weighted-\$bribery, which we state below, with a reduction from PARTITION to their single binary issue bribery problem.

Name: plurality-weighted-\$bribery.

Given: A set C of candidates. A collection V of voters specified via their preference lists (most preferred candidate), weights $h_1, \ldots, h_m \in \mathbb{Z}^+$, and prices $\$_1, \ldots, \$_m \in \mathbb{Z}^+$. A distinguished candidate $p \in C$ and $k \in \mathbb{Z}^+$ (the budget).

Question: Is there a set $B \subseteq \{1,\ldots,m\}$ such that $\sum_{i\in B} \$_i \leq k$ and there is a way to bribe the voters from B in such a way that p becomes a winner?

In fact, Faliszewski et al. showed 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 4.4.5 *Weighted-(SM,A,C)-Bribery for* $A \in \{IV, IV + DV\}$ *and* $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}\}$ *is NP-complete.*

Proof. Membership in NP is an immediate consequence of Theorem 4.4.1 and a guess and check algorithm.

We reduce the plurality-weighted-$bribery problem over two candidates to an instance of weighted-(SM,IV,C_{EQUAL})-Bribery. We construct our instance with the same number of voters ($P = V$); the same costs ($Q[i] = \$_i$); weights ($w_i = h_i$); and the same budget ($B = k$) as the original problem. In our instance we create only one issue. This issue is an independent variable in our construction and the two values for the issue correspond exactly to the two candidates. We assign the votes and p the same as in the instance of plurality-weighted-$bribery.

The instance of our problem that we construct is exactly the given instance of plurality-weighted-$bribery: our problem is a yes instance if and only if there is a solution to the original problem. Therefore, weighted-(SM,IV,C_{EQUAL})-Bribery is NP-complete.

We can extend this proof for all variants of weighted-(SM,A,C)-Bribery where $A \in \{IV, IV + DV\}$ and $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}, C_{\text{LEVEL}}, C_{\text{ANY}}\}$. Since we are only using one independent variable in the construction there is no difference between IV and (IV+DV). Likewise, since there is only one variable and therefore only one variable level, the cost scheme does not matter. ❑

We can also extend the result of Theorem 4.4.5 to the case of DV bribery.

Corollary 4.4.6 *Weighted-(SM,DV,C)-Bribery where* $C \in \{C_{\mathrm{EQUAL}}, C_{\mathrm{FLIP}}, C_{\mathrm{LEVEL}}, C_{\mathrm{ANY}}\}$ *is NP-complete.*

Proof. Membership in NP is an immediate consequence of Theorem 4.4.1 and a guess and check algorithm.

We apply the same mapping as shown in Theorem 4.4.5 to an instance of weighted-(SM,DV,C_{EQUAL})-Bribery except for the definition of the variables in our instance. Instead of only having one independent issue, we introduce an independent and dependent issue. The preference of all voters on the independent variable is $1 > 0$. We now map the preferences of the instance of plurality-weighted-\$bribery to the dependent variable when the independent variable is set to 1. We set p to be $1x$ where x is the definition of p from the instance of plurality-weighted-\$bribery. Since we are using only DV there will be no way to change the preference of any voter over the independent variable. Therefore, all bribery will take place on the dependent variable. This single dependent variable has the same candidate mapping as in the proof of Theorem 4.4.5. Therefore, there will be a solution to our constructed problem if and only if there is a solution to the given instance of plurality-weighted-\$bribery and our problem is NP-complete.

We can extend this construction to any version of Weighted-(SM,DV,C)-Bribery where $C \in \{C_{\mathrm{EQUAL}}, C_{\mathrm{FLIP}}, C_{\mathrm{LEVEL}}, C_{\mathrm{ANY}}\}$. We can set the costs of bribery on the dependent variables in each of these cost schemes to work with the above reduction.

For C_{FLIP} we set the entries of Q to be the same as in the construction for C_{EQUAL}. Since there is only one available bribery action for each voter, this will make the prices equivalent. We can set the costs in the same way for C_{ANY}. For C_{LEVEL}, each of the variables will be at level 2 (since they are all dependent on the independent issue). Therefore, we can set the entries of Q in the same way as for the other cost schemes. ❑

However, if we use the cost scheme C_{FLIP} and we assume that the bribing costs are the same for all voters, the bribery problem becomes easy.

Theorem 4.4.7 *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 $\vec{Q}$ are all the same.*

Proof. We can use the same argument as in the proof of Theorem 4.4.3. 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} pricing function, the cost to bribe any voter at any particular level is equal for all voters. We can therefore bribe the voters from heaviest to lightest (in terms of the individual voters' weight). This will be an optimal sequence of bribes at each individual level and lead to the optimal bribery scheme overall.

This is true no matter which bribery actions are allowed, since using IV or DV just restricts the issues over which it is possible to perform a flip in the CP-nets.

❏

4.4.3 One-Step Rules

In this section we will investigate the three one-step rules that we have introduced in this chapter: OP, OV, and OK. We will consider each of these rules in turn and we will see that these rules pose more significant challenges then the sequential rules. A summary of our results can be found in Table 4.2.

One-step Plurality

Recall the MINIMUM COST FEASIBLE FLOW problem introduced in Section 2.1.2 [1]. Faliszewski [51] shows that plurality bribery in single issue elections with nonuniform cost functions is in P through the use of flow networks. We showed a similar algorithm in Theorem 3.2.17 for finding possible winners with bribery in round robin sports tournaments. Both of these algorithm require the enumeration of all possible elements of the candidate

Table 4.2: Bribery and complexity results for one-step rules. OP(A) stands for voting rule OP with bribery actions A, and similarly for OV and OK, OK* stands for OK when k is a power of 2. In some cases we have not been able to provide lower bounds. In these cases we note the upper bounds with $\in$.

	C_{EQUAL}	C_{FLIP}
OP (IV)	P (Thm. 4.4.8)	P (Thm. 4.4.8)
OP (DV, IV+DV)	P (Thm. 4.4.8)	P (Thm. 4.4.8)
OV (IV)	P (Thm. 4.4.12)	P (Thm. 4.4.12)
OV (DV, IV+DV)	P (Thm. 4.4.12)	P (Thm. 4.4.12)
OK*(IV)	P (Thm. 4.4.15)	P (Thm. 4.4.15)
OK*(DV, IV+DV)	P (Thm. 4.4.15)	P (Thm. 4.4.15)
	C_{LEVEL}	C_{ANY}
OP (IV)	P (Thm. 4.4.9)	$\in$ NP (Thm. 4.4.10)
OP (DV, IV+DV)	$\in$ NP (Thm. 4.4.10)	$\in$ NP (Thm. 4.4.10)
OV (IV)	P (Coro. 4.4.13)	$\in$ NP (Thm. 4.4.10)
OV (DV, IV+DV)	$\in$ NP (Thm. 4.4.10)	$\in$ NP (Thm. 4.4.10)
OK*(IV)	P (Coro. 4.4.16)	$\in$ NP (Thm. 4.4.10)
OK*(DV, IV+DV)	$\in$ NP (Thm. 4.4.10)	$\in$ NP (Thm. 4.4.10)

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.

Theorem 4.4.8 *(OP,A,C)-Bribery, where $A \in \{IV, DV, IV+DV\}$ and $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}\}$, 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 [51]. Therefore, we assume we have a number of candidates that is superpolynomial in the number of voters. We show that a similar technique using flow networks as in [51] can be used in the superpolynomial case while still assuring a polynomial algorithm.

We can build our flow network without listing all candidates. Let candidate c_i be **safe** if $score(c_i) \leq score(p) - 1$. The key insight is that, at most n of the candidates will receive votes. For each voter v_i, we only need to consider the next n safe candidates enumerated in cost order. This is because, in the worst case, we will have to redistribute the votes of all n

voters to candidates other than those currently receiving votes. We can safely assume that we will redistribute the votes as cheaply as possible, so we need only consider, for each voter, the $\leq n$ unsafe candidates (those receiving votes) and then at most n more.

We then construct a series of networks, one for each $r \in \{1,\ldots,n\}$. We want to know whether, for any such r, p can be made a winner with exactly r votes without exceeding B. If so, then the answer to the bribery question is yes. Otherwise, we reject. We show that, for each r, the network can be constructed in time polynomial in the number of voters. Since there are only n values of r of interest, this shows that the whole (Turing) reduction can be performed in P.

To solve the decision problem for a given r, we transform this problem to a minimum-cost flow problem [1] (and discussed in Section 2.1.2). In our graph, units of flow represent votes and the minimum cost of the graph flow is the minimum cost of bribery necessary to elect p with exactly r votes.

We describe the construction similarly to Theorem 3.2.17 in Chapter 3. The network has a source s, a sink t, and two internal "layers" of nodes. We will describe the construction, moving through these three internal layers in turn.

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

The second layer models the profile modified by the bribery, where each voter can maintain or change his current 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 through bribery since either they cost too much or they already receive too many votes. Providing n safe candidates for each voter is enough, since there are n voters and in the worst case each of them votes for a different candidate. In the worst case there are $n-1$ unsafe candidates before we encounter the first safe candidate. Once we reach the first safe candidate we enumerate $n-1$ more candidates. Therefore, we only need to enumerate a polynomial number of candidates.

The second layer of nodes represents the subset of the candidates that v_i can make his top candidate. For each voter v_i, we define a subset C_i of candidates consisting of p, the candidate that v_i currently has as his top outcome, all candidates that currently receive a vote (at most n), and the n safe, cheapest candidates for this voter, according to the cost scheme, for a total of at most $2n+1$ candidates. The second layer of nodes in the network is the union, S, of all the C_is. The set S contains p and all $\leq n$ top-choice candidates, plus at most n candidates for each of the n voters. Thus, $|S| \leq (n^2+n+1)$.

For each voter node v_i in the first layer, we build an edge to every node in that voter's set of considered candidates in S. The edge from v_i to c_j has capacity $+\infty$ and cost equal to the cost to bribe v_i to vote for c_j.

Finding the candidates to include for each voter and the cost of bribery for the particular voter/candidate pair takes polynomial time no matter the cost scheme. Finding the voted candidates is easy since finding the optimal outcome in acyclic CP-nets takes polynomial time. In order to find the n cheapest safe candidates for C_{EQUAL} we start from the optimal outcome for each voters' CP-net and move down in a linearization of the outcome ordering. This method takes polynomial time, as shown by Brafman et al. [17]. 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. For C_{EQUAL} the cost is always 1. For the voted candidates, including p, it is easy to compute the cost if we use C_{FLIP} as shown in Theorem 4.4.2.

To enforce the constraint that no candidate besides p can receive more than r votes we build an edge from each candidate to the sink with 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 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 [51], 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 algorithm. By including only the cheapest n alternative candidates for each voter, along with p and all the voted candidates, the result still holds. Assume there is a minimal flow in the larger network which goes through one of the nodes which we omit. This would mean that a voter has been forced to vote for another, more expensive, non-voted candidate since all its cheapest candidates already had r votes each. But, 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 build, at worst, n networks with $O(n^2)$ nodes and edges. Since min cost feasible flow problem can be solved in $\mathscr{O}(|V||E|\log(|V|))$, the overall running time of this method is polynomial in the size of the CP-nets and n. ❑

Leveraging the above construction, we get the following theorem.

Theorem 4.4.9 *(OP,IV,C_{LEVEL})-Bribery is in P.*

Proof. The result of Theorem 4.4.8 also holds when we use the cost schemes C_{LEVEL} or C_{ANY} with the IV bribery criteria.

When not all variables are independent and we use IV, it may be impossible to find n alternative candidates for a voter, since that voter may be able to vote for fewer than n candidates given the restrictions on the bribery actions. In this case we provide all possible alternative candidates for the voter (since we may have to shuffle his vote).

For any set of voters we only need to be able, in polynomial time, to enumerate the next n possible votes in cost order. If we can achieve this enumeration then we are guaranteed, by use of the pigeon hole principal, that all voters can be moved as in Theorem 4.4.8. For the C_{LEVEL} criteria we can enumerate the next safe candidate in polynomial time. Notice that, on a per agent basis, all independent variables will have the same bribing cost. Specifically, for some agent i, the cost to make any bribery action on any single independent variables will be $k \times Q[i]$, where k is the number of levels in agent i's CP-net. This means that, when only IV is used, C_{LEVEL} is equivalent to C_{FLIP}.

Now that we can enumerate the next candidate in cost order on a per agent basis, we directly use the construction in Theorem 4.4.8. Therefore, (OP,IV,C_{LEVEL})-Bribery is in P. ❑

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 4.4.1 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 4.4.10 *(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 $D \in \{OP, OV, OK\}$.*

The following theorem leads us to the conjecture that all the problem variants listed in Theorem 4.4.10 are hard when using C_{ANY}. Recall the SUBSET SUM PROBLEM statement from Garey and Johnson [65].

Name: SUBSET SUM

Given: A set $H \in \mathbb{Z}_+$, $h_1, \ldots, h_k$, and a target $S \in \mathbb{Z}_+$.

Question: Does there exist a subset $I \subseteq \{1, \ldots, k\}$ such that $\sum_{i \in I} h_i = S$?

Using this problem we can show the following.

Theorem 4.4.11 *Computing the n cheapest next candidates with C_{ANY} is NP-hard.*

Proof. We map a given SUBSET SUM instance (H,S) in the problem of finding the next cheapest voter in an instance of (D,A,C_{ANY})-Bribery. Consider a domain with $m = k$ binary features. We set the cost m_i for some voter v_i on feature f_i to be h_i, his current vote to 0^m, and set the budget B to S. If we were to enumerate v_i's next n cheapest candidates, we will need to find a candidate such that the cost of all the flips will be exactly $B = S$. In order to perform this enumeration we need to solve an NP-complete sub-problem. Therefore computing the n cheapest next candidates with C_{ANY} is NP-hard. ❑

Theorem 4.4.11 does not clarify the results for C_{LEVEL} and the dependent variable bribery schemes. These problems create an instance similar to C_{ANY}, however, the price structure for C_{LEVEL} must be strictly decreasing as we go down levels. This does not give us the freedom to obviously embed a SUBSET SUM instance like we constructed in the proof of Theorem 4.4.11. We are continuing to study the complexity of problems involving C_{LEVEL} and dependent variable bribery.

One-step Veto

The OV provides a particular set of challenges as noted in Theorem 4.4.1 because we cannot necessarily enumerate the entire winning set. However, as we show, bribery with the OV rule is still computationally easy in most cases.

Theorem 4.4.12 *(OV,A,C)-Bribery is in P if* $A \in \{IV, DV, IV+DV\}$ *and* $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}\}$

Proof. The choice of IV, DV, or IV+DV, does not matter in this context. We can either change a voters' last place candidate or we can't, the method doesn't matter.

We note that to compute the worst candidate for every voter it is sufficient to reverse the orderings in all the cp-statements as discussed in Section 4.1.2. Let us call any CP-net on which this is performed a reversed CP-net. If p has no vetoes then we trivially accept (since he is, at worst, tied for a winning position). Otherwise we break into two cases:

when $n < 2^m$ 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. When $n \geq 2^m$ we consider the reversed CP-nets and we use an algorithm based on a flow network similar to the one in Theorem 4.4.8 to redistribute 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 network is the same as the one in Theorem 4.4.8 except that nodes not representing p have capacity ∞ into the sink. ❑

Additionally, we can state the following corollary similar to Section 4.4.3. The proof is a straightforward combination of Theorem 4.4.9 and the observation in Theorem 4.4.12.

Corollary 4.4.13 *(OV,IV,C_{LEVEL})-Bribery is in P.*

One-step k-Approval

In One-step k-approval (OK) we aggregate the n profiles by taking the top k outcomes from each agents' profile. In a CP-net, the top outcome is followed by a series of outcomes obtained through a sequence of worsening flips. Since CP-nets induce a partial order, we mean the top k according to some linearization. Here we assume that the linearization is according to variable order and then lexicographically. Other linearizations would produce the same results. 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 [52]. However, when agents' preferences are expressed as CP-nets, bribery schemes for k-approval can be computed in polynomial time under certain conditions. First we need a lemma.

Lemma 4.4.14 *Given an acyclic CP-net X and a constant linearization scheme across all agents and $k = 2^j$, 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 the following proof, is for the elicitation problem for CP-nets: if the ordering over the issues is known, then we only need to elicit the top candidate in order to know a voters' top k candidates, when k is a power of 2.

Theorem 4.4.15 *When $k = 2^j$, (OK,A,C)-Bribery is in P if $A \in \{IV, DV, IV + DV\}$ and $C \in \{C_{\text{EQUAL}}, C_{\text{FLIP}}\}$,*

Proof. When k is a power of two Lemma 4.4.14 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 (group of candidates that are all approved together). For each voter we will need to find n cheapest un-voted bundles. We can do this in polynomial time as all we require is the ability to apply *next* some multiple of k times. We then apply the algorithm in the proof of Theorem 4.4.8 in order to decide the cheapest bribery scheme to elevate the bundle that includes p into the winning set. ❑

Again, similar to our results for OV, we can state the following corollary.

Corollary 4.4.16 *When $k = 2^j$, (OK,IV,C_{LEVEL})-Bribery is in P.*

4.4.4 Manipulation and Non-binary Domains

We will now show that, in the setting we have considered, manipulation is easy for two of the voting procedures.

Theorem 4.4.17 *Weighted-CCM and unweighted-CCM are in P for SM, OP, OV, and OK (when $k = 2^j$).*

Proof. Note that in the CCM problems (weighted or unweighted) we are given two sets of voters, X and Y. We are asked to set the CP-nets of Y in such a way as to ensure a victory for a candidate p. All the voters in Y have completely unspecified CP-nets.

When voting with SM and OP the optimal thing to do is to set the CP-nets of voters in Y in such a way that p is their optimal outcome. This can be done in polynomial time. If p can win with all the votes from Y then we accept, otherwise we reject.

When voting with OV we can count how many vetoes p has in the set X. If p currently receives no vetoes then we set the votes of Y to veto any arbitrary candidate that is not p. If the total number of vetoes p receives in X is such that we can evenly distribute the votes of Y among the other candidates so that every candidate has as many or more vetoes than p then we can make p a winner. We can do this in polynomial time since we must check each voter's veto and the number of voters is polynomial. However, if the number of vetoes for p is greater than the number we can apply to all other candidates, then we cannot make p a winner.

For OK when $k = 2^j$ we can leverage the observation in Lemma 4.4.14. In this case, as in the case for SM and OP, we set all the voters in Y in such a way that p is the top candidate. Since we have no control over the next candidates this is the optimal action.

If p can win with all the votes from Y assigned in the above ways then we accept, otherwise we reject. ❑

Up to this point we have required that all variables in the domain be binary. While this is a convenient and highly expressive model, we wish to relax this assumption. This allows the sequential composition of general voting rules (not just majority) as in other work on voting in sequential domains [82]. This allows for interesting extensions in our domain and may lead to some novel insights about bribery and manipulation in sequential domains when agent's preferences are represented as CP-nets.

In what follows, each issue has a domain with two or more values and the CP tables specify strict total orderings on such domains. For each variable $x \in M$, we associate a domain $D(x)$ of values to that variable. When using the DV bribery action, we fix some part of the linear order. The briber will modify the linear orders through a sequence of swaps, much like in the swap bribery domain described by Elkind et al. [43]. Note that editing dependencies could change the linear order on a particular level.

Now we can use different voting rules at every level. We associate to a profile (P, O) a vector R of voting rules $\langle r_1, \ldots, r_m \rangle$. At level i we use voting rule r_i to aggregate the preferences at that level and propagate the winning outcome at that level to the next level for all agents' CP-nets. This is similar to Della Pozza et al.'s [37] definition for sequential voting rules with multivariate domains with soft constraints. Winner determination in this domain is polynomial if evaluation of the underlying voting rules are polynomial procedures and we can use the same algorithm as in Theorem 4.4.1. With this model we can show some additional theorems about sequential voting rules.

Theorem 4.4.18 *If bribery or CCM is computationally hard for any rule in $\langle r_1, \ldots, r_m \rangle$ then bribery is hard for the sequential rule.*

Proof. This property can be seen from a direct example. Suppose that we had a voting scenario with two levels: the first to be aggregated by any easy rule shown in this section and the second level by any hard rule (such as the Borda rule [12, 38]). Since we are using a sequential rule, the first level will be computationally easy to manipulate since we can ignore effects on the variables of the second level. However, the second level will remain computationally difficult to manipulate. Therefore, the inclusion of any hard rule in the sequential rules R makes bribery hard for the overall scenario. ❑

Theorem 4.4.19 *If bribery or CCM is easy for every rule in $\langle r_1, \ldots, r_m \rangle$ then bribery is easy for the sequential rule.*

Proof. This follows from the separability property shown in Theorem 4.4.3. Since the levels of the ordering O can be treated independently, the problem separates into a sequence of independent bribery problems. If each one of these sub-problems are computationally easy, the entire sequence is computationally easy because we must win at every level. ❑

4.5 Observations

The results of our study are summarized in Table 4.1 and Table 4.2. We note that not all our results are in these tables, since some of the proofs of properties and interesting sub-cases do not neatly fit into tabular form. Additional results not appearing in our tables are our results for complexity of the CCM problem in sequential domains and our results about sequential rules with non-binary issues.

We show that SM bribery is easy except when we use C_{EQUAL}. However, when voters are weighted, bribery for SM is almost always difficult. OP and OV bribery is easy except for the two cases we have C_{LEVEL} with any variant of DV or any case with C_{ANY}. In these cases, we do not have a formal result although we believe the problem is difficult. Bribery is easy also for OK when k is a power of 2. This is particularly surprising at first sight, since bribery for k-approval in a non-combinatorial setting is difficult.

We have also shown several 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.

4.6 Summary

In this chapter we have studied the complexity of bribery and manipulation problems in elections with combinatorially structured domains where agents' express their preferences as CP-nets. To perform this study we introduced four cost schemes, three bribery schemes, and two broad classes of voting rules for combinatorial domains. This chapter represents novel work in that we are the first to extend the bribery and manipulation problems to combinatorially structured domains. In general we find that bribery in these settings is computationally easy for all the non-weighted sequential rules and many of the one-step rules. We are continuing to extend this work with studies of additional voting rules for the one-step instances, allowing domains with non-binary variables, and investigating the possibility of the bribery actor being able to add conditional preferences to the individual agents.

The complexity questions we have answered in this chapter are unsettling. We have seen that it may be computationally easy to affect elections in domains of these types. At first glance, this is odd since we have the possibility of an exponential number of candidates with respect to the size of the problem input. However, as we have seen, we can effectively work around this possible computational barrier. Our results may be due to an over-constrained model and our continuing work will address this issue.

In this chapter and the previous chapter we have focused on the theoretical implications of the security of election systems. We have established some baselines as to the computational complexity of reasoning in a variety of bribery domains. However, we are left with the underlying question of whether or not our results are applicable in actual elections (either involving people or multiagent systems). We dive into this question of practicality with the next chapter that looks at elections with real data and seeks to understand the empirical implications of our theoretical results.