Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Bribery in Voting With Soft Constraints

Maria Silvia Pini[1] and **Francesca Rossi**[2] and **Kristen Brent Venable**[3]
[1]: Department of Information Engineering, University of Padova, Italy
e-mail: pini@dei.unipd.it
[2]: Department of Mathematics, University of Padova, Italy
e-mail: frossi@math.unipd.it
[3]: Department of Computer Science, Tulane University and IHMC, USA
e-mail: kvenabl@tulane.edu

Abstract

We consider a multi-agent scenario where a collection of agents needs to select a common decision from a large set of decisions over which they express their preferences. This decision set has a combinatorial structure, that is, each decision is an element of the Cartesian product of the domains of some variables. Agents express their preferences over the decisions via soft constraints. We consider both sequential preference aggregation methods (they aggregate the preferences over one variable at a time) and one-step methods and we study the computational complexity of influencing them through bribery. We prove that bribery is NP-complete for the sequential aggregation methods (based on Plurality, Approval, and Borda) for most of the cost schemes we defined, while it is polynomial for one-step Plurality.

We consider a multi-agent scenario where a collection of agents needs to select a decision from a large set of decisions, over which they express their preferences. This set has a combinatorial structure, i.e., each decision is the combination of certain features, where each feature has a set of possible instances. This occurs in many AI applications, such as combinatorial auctions, web recommender systems, and configuration systems.

Even if the number of features and instances is small, the number of possible decisions can be very large. However, agents may describe their preference in a compact and efficient way, using a preference modelling/reasoning formalism such as soft constraints (Bistarelli, Montanari, and Rossi 1997), CP-nets (Boutilier et al. 2004), or graphical utility models (Bacchus and Grove 1995). In this paper we consider fuzzy and weighted soft constraints.

To model preference aggregation, we consider the use of voting rules (Arrow and K. Suzumura 2002), following two approaches: a sequential one, where agents vote on each feature at a time, and a one-step approach, where agents vote just once over decisions regarding all features.

All the ways to aggregate the preferences of several agents into one collective decision are subject to attempts of various entities to influence the result for their good. Single agents, or a collection of them, may try to manipulate the voting rule by misreporting their preferences (Gibbard 1973). Also, the voting chair can control some of the parameters of the voting rule (candidates, voters, etc.) (Bartholdi, Tovey, and Trick 1992). Finally, an external agent may try to convince some of the voters to change their preferences in order to get a collective decision which is more preferred to him (Faliszewski, Hemaspaandra, and Hemaspaandra 2009). In this paper we focus on this third way to influence the result, usually called *bribery*, and we study its computational complexity. Voting rules where bribery is computational complex can be considered resistant to bribery. More precisely, the question we address is the following one: How computationally complex is it for the briber to determine whether by paying certain agents to change their preferences, within a certain budget, a specified candidate can be made the winner decision? We measure the computational complexity of the bribery problem, thus assuming that a computationally complex bribery problem makes the aggregation resistant to bribery.

In this paper we define several cost schemes to compute the cost for an agent to satisfy a briber's request. We show that the sequential approaches (which are based on voting rules such as Plurality, Approval, and Borda), are all resistant to bribery for most of the cost schemes, while one-step Plurality is instead vulnerable to bribery (that is, bribery is computationally easy).

Many papers on bribery assume that the agents express their preferences over a small set of candidates (Faliszewski, Hemaspaandra, and Hemaspaandra 2009; Faliszewski 2008; Elkind, Faliszewski, and Slinko 2009; Faliszewski et al. 2007). We assume instead agents express compactly their preferences over a large set of candidates with a combinatorial structure. Bribery when agents vote over a large set of candidates has been considered also in (Mattei et al. 2012b; 2012a; 2013), but preferences were modeled via CP-nets. and not via soft constraints. Voting with soft constraints has been also studied in (Dalla Pozza, Rossi, and Venable 2011; Dalla Pozza et al. 2011), but bribery was not investigated there.

The paper is a revised and extended version of (Pini, Venable, and Rossi 2013).

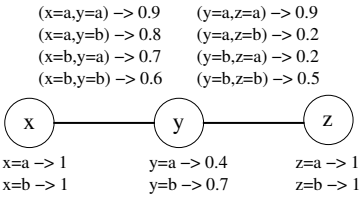
Background

Soft Constraints

A soft constraint (Bistarelli, Montanari, and Rossi 1997) involves a set of variables and associates a value from a (partially ordered) set to each instantiation of its variables. This value is taken from a preference structure (called a c-semiring) defined by $\langle A, +, \times, 0, 1\rangle$, where A is the set of preference values, $+$ induces an ordering over A (where $a \leq b$ iff $a + b = b$), $\times$ is used to combine preference values, and 0 and 1 are resp. the worst and best preference value. A Soft Constraint Satisfaction Problem (SCSP) is a tuple $\langle V, D, C, A\rangle$ where V is a set of variables, D is the domain of the variables, C is a set of soft constraints (each one involving a subset of V), and A is the set of preference values.

An instance of the SCSP framework is obtained by choosing a specific preference structure. For instance, a classical CSP (Rossi, Van Beek, and Walsh 2006) is just an SCSP where the structure is $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. Hence, SCSPs generalize CSPs. Choosing $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$ instead means that preferences are in [0,1] and we want to maximize the minimum preference. This is the setting of fuzzy CSPs (FCSPs), that we will use in the following examples. We will also consider the setting of weighted CSPs (WCSPs), where the structure is $S_{WCSP} = \langle R^+, min, +, +\infty, 0\rangle$, i.e., preferences are interpreted as costs from 0 to $+\infty$, and we want to minimize the sum of the costs.

The figure below shows the constraint graph of an FCSP where $V = \{x, y, z\}$, $D = \{a, b\}$ and $C = \{c_x, c_y, c_z, c_{xy}, c_{yz}\}$. Each node models a variable and each arc models a binary constraint, while unary constraints define variables' domains. For example, c_y associates preference 0.4 to $y = a$ and 0.7 to $y = b$. Default constraints such as c_x and c_z will often be omitted in the following examples.



Solving an SCSP means finding some information about the ordering induced by the constraints over the set of all complete variable assignments. In the case of FCSPs and WCSPs, such an ordering is a total order with ties. In the example above, the induced ordering has ($x = a$, $y = b$, $z = b$) and ($x = b$, $y = b$, $z = b$) at the top, with preference 0.5, ($x = a$, $y = a$, $z = a$) and ($x = b$, $y = a$, $z = a$) just below with 0.4, and all others tied at the bottom with preference 0.2. An optimal solution, say s, of an SCSP is then a complete assignment with an undominated preference (thus ($x = a$, $y = b$, $z = b$) or ($x = b$, $y = b$, $z = b$) in this example).

While (soft) constraint problems are in general NP-complete (Rossi, Van Beek, and Walsh 2006), constraint problems where the connectivity graph has the form of a tree are polynomial to solve (Dechter 2006). In general constraint problem are solved via search. However, constraint propagation (which is a sort of inference that may reduce the variables' domains) may help the search for an optimal solution. Given a variable ordering o, an FCSP is directional arc-consistent (DAC) if, for any two variables x and y linked by a fuzzy constraint, such that x precedes y in the ordering o, we have that, for each a in the domain of x, $f_x(a) = max_{b \in D(y)}(min(f_x(a), f_{xy}(a, b), f_y(b)))$, where f_x, f_y, and f_{xy} are the preference functions of c_x, c_y and c_{xy}. This definition can be generalized to any SCSP instance by replacing max with $+$ and min with $\times$. Thus, for WCSPs max must be replaced with min and min with sum. DAC applied bottom-up on the tree shape of the problem is enough to find the preference level of an optimal solution when the problem has a tree-shaped constraint graph and the variable ordering is compatible with the father-child relation of the tree. In fact, the optimum preference level is the best preference level in the domain of the root variable. The tree-like restriction is not the only one to assure tractability. Instead of a tree, we can have a graph with cycles but with a bounded treewidth. Many classes of graphs have a bounded treewidth (Bodlaender 1998; Thorup 1998).

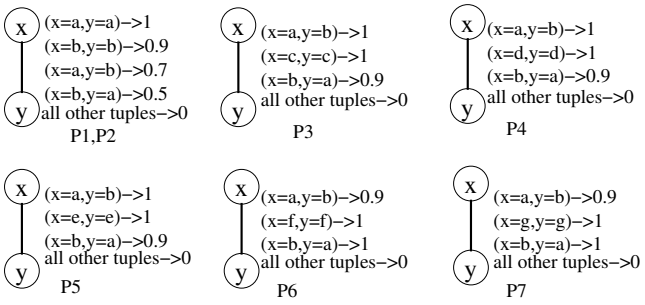
Voting Rules

A voting rule allows a set of voters to choose one among a set of candidates. Voters need to submit their vote, that is, their preference ordering (or part of it) over the set of candidates, and the voting rule aggregates such votes to yield a final result, usually called the winner. In the classical setting (Arrow and K. Suzumura 2002), given a set of candidates C, a *profile* is a collection of total orderings (or parts of them) over the set of candidates, one for each voter. Given a profile, a *voting rule* maps it onto a single winning candidate (if necessary, ties are broken appropriately). In this paper, we will often use a terminology which is more familiar to multi-agent settings: we will sometimes call the voters "agents" , the candidates "solutions", and the winning candidate "decision" or "best solution". Some widely used voting rules, that we will study in this paper, are: *Plurality*, where each voter states a single preferred candidate, and the candidate who is preferred by the largest number of voters wins; *Borda*, where given m candidates, each voter gives a ranking of all candidates, the i^{th} ranked candidate gets a score of $m - i$, and the candidate with the greatest sum of scores wins, and *Approval*, where given m candidates, each voter approves between 1 and $m - 1$ candidates, and the candidate with most votes of approval wins.

Sequential Preference Aggregation

Assume to have a set of agents, each one expressing its preferences over a common set of objects via an SCSP whose variable assignments correspond to the objects. Since the objects are common to all agents, all the SCSPs have the same set of variables and the same variable domains but they may have different soft constraints, as well as different preferences over the variable domains. In (Dalla Pozza,

Rossi, and Venable 2011) this is the notion of *soft profile*, which is a triple (V, D, P) where V is a set of variables (also called issues), D is a sequence of $|V|$ lexicographically ordered finite domains, and P a sequence of m SCSPs over variables in V with domains in D. A soft profile consists of a collection of SCSPs over the same set of variables, while a profile (as in the classical social choice setting) is a collection of total orderings over a set of candidates. A *fuzzy profile* (resp., *weighted profile*) is a soft profile with fuzzy (resp., weighted) soft constraints. An example of a fuzzy profile where $V = \{x, y\}$, $D_x = D_y = \{a, b, c, d, e, f, g\}$, and P is a sequence of seven FCSPs, is shown in the figure below.



The idea proposed in (Dalla Pozza, Rossi, and Venable 2011) to aggregate the preferences in a soft profile in order to compute the winning variable assignment is to sequentially vote on each variable via a voting rule, possibly using a different rule for each variable. Given a soft profile (V, D, P), assume $|V| = n$, and consider an ordering of the variables $O = \langle v_1, \ldots, v_n \rangle$ and a corresponding sequence of local voting rules $R = \langle r_1, \ldots, r_n \rangle$. The sequential procedure is a sequence of n steps, where at each step i, we perform the following tasks. All agents are asked for their preference ordering over the domain of variable v_i, yielding profile p_i over such a domain. To do this, the agents achieve DAC on their SCSP, considering the ordering O. Then, the voting rule r_i is applied to profile p_i, returning a winning assignment for variable v_i, say d_i. If there are ties, the first one following the given lexicographical order will be taken. Finally, the constraint $v_i = d_i$ is added to the preferences of each agent and DAC is achieved to propagate its effect considering the reverse ordering of O.

After all n steps have been executed, the winning assignments are collected in the tuple $\langle v_1 = d_1, \ldots, v_n = d_n \rangle$, which is declared the winner of the election. This is denoted by $Seq_{O,R}(V, D, P)$. An example of how the sequential procedure works can be found in (Dalla Pozza et al. 2011). A similar sequential procedure has been considered in (Lang and Xia 2009), when agents' preferences are expressed via CP-nets.

Winner Determination

We consider two main approaches: sequential and one-step. For the sequential approach, we employ the sequential procedure described above with Plurality, Approval and Borda rules. We have an ordering O over the variables and we consider each variable in turn in such an ordering. At each step, each agent provides some information about the considered variable, say X, which depends on the voting rule we use: in *Sequential Plurality* (SP) every agent provides one best value for X, in *Sequential Approval* (SA) all best values for X, while in *Sequential Borda* (SB) a total order (possibly with ties) over the values of X, along with the preference values for each domain element. We then choose one value for the considered variable, as follows: with SP and SA we choose the value voted by the highest number of agents, with SB we select the value with best score, where the score of a value is the sum of its preferences over all the agents. Note that "best" means maximal in the case of fuzzy constraints and minimal in the case of weighted constraints. Once a value is chosen for a variable, this value is broadcasted to all agents, who fix X to this value in their soft constraints and achieve DAC in the reverse ordering w.r.t. O. We continue with the next variable, and so on until all variables have been handled.

The alternative to a sequential approach is a one-step approach, where each agent votes over decisions regarding all variables, not just one at a time. A possible voting rule to use is what we call *One-step Plurality* (OP), where each agent provides an optimal solution of his SCSP, and we select the solution which is provided by the highest number of agents. We don't consider one-step Approval since voting could require exponential time since each agent may have an exponential size set of optimals.

In this paper we will study the resistance to bribery for SP, SA, SB, and OP when agents' preferences are expressed via (possibly different) tree-shaped fuzzy or weighted CSPs. This is an interesting problem since winner determination for these rules is computationally easy.

Theorem 1 *Winner determination takes polynomial time for SP, SA, SB, and OP when agents' preferences are tree-shaped fuzzy or weighted CSPs.*

Proof: The fact that we are considering tree-shaped soft constraint problems ensures that voting, in all these cases, can be done in polynomial time by achieving DAC. Winner determination is then polynomial as well, since it just requires a number of polynomial steps which equals the number of variables. For OP, computing an optimal solution is polynomial on tree-shaped soft constraint problems, so voting is polynomial. Determining the winner requires just counting the number of votes for each of the voted candidates (which are in polynomial number), so it is polynomial as well. □

An approach like OP is less satisfactory than the sequential approaches in terms of *ballot expressiveness*: since the number of solutions may be exponentially large with respect to the number of agents, 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 (that is 1 for OP) to the number of solutions they can vote for. So there is trade-off between easiness of computing votes and ballot expressiveness.

The Bribery Problem

We now define formally the bribery problem in our scenario, where agents express their preferences via fuzzy and weighted soft constraints. We recall that bribery is an attempt to modify the result of the election where there is an outside agent, called the briber, that wants to affect the result of the election by paying some voters to change their votes, while being subject to a limitation of its budget. In defining bribing scenarios in our context, it is thus necessary to decide what the briber can ask an agent to do (for example, just making a certain candidate optimal, or changing more of its preference ordering) and how costly it is for the briber to submit a certain request. The cost usually represents the effort the agent has to make to satisfy the briber's request. If we use Plurality to determine the winner, either in its sequential or one-step version, the most natural request a briber can have for an agent is to ask the agent to make a certain solution (or a certain value in the sequential case) optimal in his soft constraint problem. In order to do this, the agent can modify the preference values inside its variable domains and/or constraints.

We define in several ways the cost of a briber's request, which is to make a certain solution A optimal[1]:

- C_{equal}: The cost is fixed (without loss of generality, we will assume it is 1), no matter how many changes are needed to make A optimal;
- C_{do}: The cost is the distance from the preference value of A, denoted by $pref(A)$, to the preference value of an optimal solution of the SCSP of the agent, denoted by opt. If we are dealing with fuzzy numbers and we may prefer to have integer costs, the cost is defined as $C_{do} = (opt - pref(A)) * l$, where l is the number of different preference values allowed. For example, if the fuzzy preferences have 2 digits of precision, we have 100 different preferences and we will, thus, have $l = 100$.

  With weighted constraints, where preference values are costs, $C_{do} = pref(A) - opt$, since opt is the smallest cost associated to any solution.
- C_{don}: The cost is determined by considering both C_{do} and the minimum number of preference values, say t, associated to subparts (aka tuples) of A in the constraints, that must be modified in order to make A optimal. With fuzzy constraints the cost is defined as $C_{don} = ((opt - pref(A)) * l * M) + t$, while with weighted constraints the cost is defined as $C_{don} = ((pref(A) - opt) * M) + t$, where M is a large integer which must be greater than $2n - 2$ and $1 \leq t \leq 2n - 1$, where n is the number of variables. The role of M is to ensure a higher bribery cost for a less preferred candidate. We omit the details about the bounds of M due to lack of space. The upper bound of t is $2n - 1$ since, in a tree-shaped fuzzy problem, we will have n unary soft constraints over the domains of the variables and $n - 1$ binary soft constraints. Thus each solution has $2n - 1$ subparts in the constraints.
- C_{dow}: The cost is computed similarly to C_{don}, but each preference value to be modified is associated with a cost proportional to the change required on that preference. Let us denote by t_i any tuple of A with preference $\leq opt$. For fuzzy constraints, the cost is $C_{dow} = ((opt - pref(A)) * l * M) + \sum_{t_i}(opt - pref(t_i)) * l$: where the role of M is similar to the one in C_{don} but now its lower bound now depends also on the number of preference levels. M must be greater than $l(2n-2)-1$. We omit details due to lack of space.

  For weighted constraints, we define $C_{dow} = ((pref(A) - opt) * M) + \sum_{t_i}(pref(t_i) - opt)$. However, $\sum_{t_i}(pref(t_i) - opt) = pref(A) - opt$, thus $C_{dow} = ((pref(A) - opt) * (M + 1))$. So C_{dow} induces basically the same costs as C_{do}.

These cost schemes provide a measure of the number and magnitude of local changes needed in the compact preference structure for implementing the briber's request.

With the sequential approaches to determine a winner (SP, SA, and SB), it is also reasonable to consider cost schemes where the voter charges a cost for each modification required for each variable. We consider the cost scheme C_k, which takes the minimum Kemeny distance between the ordering induced by the preferences on the domain of a variable X and any ordering implementing the request of the briber. The Kemeny distance between two orderings is the number of pairs on which they differ (Kemeny 1959). More formally, given a set of candidates Ω and two orderings $o_1 = (a_1, \ldots, a_n)$ and $o_2 = (b_1, \ldots, b_n)$ over Ω, the *Kemeny distance* between o_1 and o_2, say $k(o_1, o_2)$ is $\sum_{i,j=1,\, i<j}^{n} |(sgn(a_i - a_j) - sgn(b_i - b_j)|$.

We are now ready to define formally our bribery problem.

Definition 2 *Given a voting rule V and a cost scheme C, we denote by (V, C)-Bribery the problem of determining if it is possible to make a preferred candidate win, when voting rule V is used, by bribing agents and by spending less than a certain budget according to cost scheme C.*

Determining the cost to respond to a briber's request is easy for all cost schemes.

Theorem 3 *Given a tree-shaped fuzzy or weighted CSP and an outcome A, determining the cost to make A an optimal outcome is in P for C_{equal}, C_{do}, C_{don}, and C_{dow}.*

Proof: We can check if A is already optimal in polynomial time by first computing the optimal preference opt and then checking if it coincides with the preference of A, denoted $pref(A)$. If so, the cost is 0. Otherwise, with C_{equal} the cost is always 1. To compute the cost according to C_{do}, C_{don}, and C_{dow}, we need to compute opt, the numbers of tuples of A with preference worse than opt, and the distance of their preferences from opt. These components can be computed in polynomial time for tree-shaped problems. □

Theorem 4 *Given a tree-shaped fuzzy or weighted CSP, one of its variables X, and a set of values in the domain of X, say V_X, determining the minimum cost to make all values in V_X optimal in the domain of X takes polynomial time for C_k.*

[1] In the names of the cost schemes $_{do}$ stands for *distance* from the *optimal* preference, $_n$ for the *number of preference values* that must be changed to make A optimal, and $_w$ for *weighted* cost.

Proof: Both fuzzy and weighted preferences induce a total order with ties over the values in the domain of X. In order to make all values in V_X optimal the cheapest thing to do, according to C_k, is to change their preferences to the optimal one. This, in fact minimizes the number of inverted pairs and thus the Kemeny distance between the old and the new ordering.

Bribery results for SP, SA, and SB

Theorem 5 *(V, C)-Bribery is NP-complete (and also W[2]-complete with parameter being the budget) for $V \in \{SP, SA, SB\}$ and $C \in \{C_{equal}, C_{do}\}$.*

Proof: We provide a polynomial reduction from the OPTIMAL LOBBYING (OL) problem (Christian et al. 2007). In this problem, we are given an $m \times n$ $0/1$ matrix E and a $0/1$ vector $\vec{x}$ of length n 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. We then ask if 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 with parameter k. By giving a polynomial reduction from OL to our bribery problem, we show that our problem is NP-complete (actually $W[2]$-complete with parameter being the budget B). Given an instance $(E, \vec{x}, k)$ of OL, we construct an instance of (V-C_{do})-Bribery, where $V \in \{SP, SA, SB\}$, containing constraints with only independent binary variables. The number of variables, n, is equal to the number of columns in E. For each row of E, we create a voter with the preferences over the n variables as described in the row of E. In particular, for each variable the value indicated in the row will be associated with preference 1 while the other value will be associated with preference 0. Thus, each voter has a unique most preferred solution with preference 1 and all other complete assignments have preference 0. We set the preferred outcome $A = \vec{x}$. This means that according to C_{do}, all voters not voting for A have the same cost to be bribed, which is $(opt - pref(A)) * 2 = (1 - 0) * 2 = 2$. Finally, we set the budget $B = 2k$. With C_{equal}, the cost is always 1 if A is not already voted for. We note that since we have only two values for each variable, SP, SA and SB coincide with sequential majority, thus A 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.

Since both fuzzy and weighted CSPs generalize CSPs, the result holds also for such classes of soft constraints. □

Theorem 6 *(V,C)-Bribery is NP-complete for $V \in \{SP, SA, SB\}$ and $C \in \{C_{don}, C_{dow}\}$ if $M > n * m$, where n is the number of variables and m the number of voters.*

To prove this result we can use a reduction similar to the one described for Thm. 5 from the OPTIMAL LOBBYING PROBLEM. We omit details due to lack of space.

Theorem 7 *(SA,C_k)-Bribery is NP-complete, while (SP,C_k)-Bribery and (SB,C_k)-Bribery are in P assuming ties are broken in favor of the briber.*

Proof: The first result follows from NP-completeness of bribery for Approval in the single variable case (Faliszewski 2008; Elkind, Faliszewski, and Slinko 2009). The other results can be shown as follows. By definition of SP and SB, an outcome wins if and only if its projection on each variable is the winner of the local election. After a local election, the result is fixed and propagated in all the SCSPs. Thus, it is enough to prove that bribery is in P at each step. Bribery with Plurality has been shown to be in P in (Faliszewski 2008) even with non-uniform costs. This, together with the fact that computing the cost is polynomial for C_k (Theorem 4), and that ties are broken in favor of the briber, allows us to conclude for SP. A similar proof works for Borda. □

Bribery results for OP

Bribery is computationally easy for OP. To show this we need to compute n cheapest alternative candidates for an agent to vote for. We first study this task. We assume that each agent has a linear ordering over the variables.

Theorem 8 *Computing a set of k cheapest outcomes is in P, in a tree-shaped fuzzy CSP according to C_{equal}, C_{do}, C_{don}, and C_{dow} in a weighted CSP according to C_{do}, C_{dow}, and C_{equal}, when k is given in unary.*

Proof: The cost of an outcome according to C_{do} is an integer proportional to the distance between the preference of the outcome and the preference of an optimal outcome. In order to compute k cheapest solutions, we assume to have a linear order over the variables and the values in their domains. For tree-shaped fuzzy CSPs, it has been shown in (Brafman et al. 2010) that, given such linear orders and an outcome s, it is possible to compute, in polynomial time, the outcome following s in the induced lexicographic linearization of the preference ordering over the outcomes. The procedure that performs this is called Next. Thus, in order to compute k cheapest according to C_{do}, we compute the first optimal outcome according to the linearization and then we generate the set of k cheapest candidates by applying Next $k-1$ times (each time on the outcome of the previous step). Similarly, computing the k best solutions of a weighted CSP can be done in polynomial time by using the procedure suggested in (Rollon, Flerova, and Dechter 2011). The result for C_{dow} in weighted CSPs follows immediately from the fact that, for weighted CSPs, C_{dow} is proportional to C_{do}. If we consider C_{equal}, an agent will not charge the briber for changing his vote to another optimal candidate and will charge a fixed cost to change his vote in favor of any other (non-optimal) candidate. Thus any of the above procedures can be used (although, if k exceeds the cardinality of the set of optimal solutions, the remaining ones could, in principle, be generated randomly in a much faster way). For C_{don} and C_{dow} in fuzzy CSPs, the proof is based on a polynomial time algorithm, $KCheapest$. It takes in input a tree-shaped fuzzy CSP P, an integer k, and a cost scheme C. It returns a set of k solutions of P. It first finds k optimal solutions of P, or all optimal solutions if they are less than k. If the number of solutions found is k, it stops, otherwise it looks for the remaining top solutions within non-optimal solutions by using an auxiliary weighted CSP (omitted for lack of space)

where the weights are defined by considering the adopted cost scheme. By construction, the returned solutions of the algorithm are the k-cheapest according to the selected cost scheme (or all the solutions if the k exceeds the total number of solutions) and thus we conclude. □

To show that bribery with OP is easy, we adapt to the case of an exponential number of candidates the proof shown in (Faliszewski 2008) for bribery when voting with plurality in single variable elections with non-uniform cost schemes. The algorithm requires the enumeration of all candidates 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 works by considering only a polynomial number of candidates.

Theorem 9 *(OP,C)-Bribery is in P for $C \in \{C_{equal}, C_{do}, C_{don}, C_{dow}\}$ when agents vote with tree-shaped fuzzy CSPs and for $C \in \{C_{equal}, C_{do}, C_{dow}\}$ when agents vote with tree-shaped weighted CSPs.*

Proof: We consider all $r \in \{1, \ldots, n\}$ and ask if the bribers' favorite candidate A can be made a winner with exactly r votes without exceeding its budget B. If there is at least one r such that this is possible, then it means that the answer to the bribery problem is yes, otherwise it is no. 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 (Ahuja, Magnanti, and Orlin 1993). The network has a source s, a sink t, and three "layers" of nodes.

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.

For the second layer of nodes, for each voter in the given profile, we add in this second layer nodes corresponding to A, 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, thus adding at most $2n + 1$ candidates for each voter. Intuitively, this second layer models the profile modified by the bribery, where each voter can change its vote or also maintain the previous one. The important point is that the non-voted candidates that we do not include in the second layer can be avoided since not using them does not increase the cost of the bribery. Providing n non-voted candidates for each voter is enough, since there are n voters and in the worst case each of them has to vote for a different candidate. For each node S_{ij} in the second layer corresponding to voter v_i, we add an edge from v_i to S_{ij} with capacity $+\infty$ and cost equal to the cost of bribing v_i to vote for the candidate corresponding to node S_{ij}. Finding such candidates, and the cost for the voter to vote for them, takes polynomial time, no matter the cost scheme. Finding the voted candidates is easy since finding the optimal outcome in tree-shaped fuzzy or weighted CSPs takes polynomial time. Finding the n cheapest non-voted candidates, can be done by applying the procedures described in previous section. In general, it is sufficient to compute the $2n$ cheapest candidates in order to make sure we have at least n non-voted candidates. Moreover, given a voter, computing the cost for such a voter to vote for one of the candidates is easy for both voted and non-voted candidates as shown in Theorem 3.

In the third layer of the network, we add a node for each candidate who already appears somewhere in the network (up to $n^2 + n + 1$). One of these nodes represents A. These third layer nodes are the nodes that enforce the constraint that no candidate besides A can receive more than r votes. 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 A to the sink, while for all other candidates it is a large integer M to force as much flow through the node A 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 (Faliszewski 2008), 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 A and all the voted candidates, the result still holds. In fact, assume there is a minimum-cost 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 its 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 $O(n^3)$ edges. Since minimum-cost feasible flow problem can be solved in polynomial time in the number of nodes and edges using for example the Edmonds-Karp algorithm (Ahuja, Magnanti, and Orlin 1993), the overall running time of this method is polynomial. □

Conclusions and Future Work

We defined the bribery problem when agents vote with tree-shaped fuzzy or weighted CSPs introducing several cost schemes to evaluate the cost of the bribery's request. We studied the resistance to bribery for sequential procedures (SP, SA, and SB) and one-step Plurality (OP) according to these cost schemes. Our complexity results are summarized in Table 1. It is clear that sequential approaches should be preferred. Moreover, when a problem is polynomial for SCSPs, it is also so for CSPs. Thus, OP is easy to bribe when agents vote with CSPs, and the same holds for SP and SB with C_k. In this paper we considered some widely used voting rules, that take preference orderings as input, so the numerical information contained in the soft constraints is not exploited. We plan to investigate voting rules that exploit also such information, as well as the applicability of the bribery results in preference optimization and compilation.

	SP	SA	SB	OP
C_{equal}	NP-c	NP-c	NP-c	P
C_{do}	NP-c	NP-c	NP-c	P
C_{don}	NP-c*	NP-c*	NP-c*	P/ ?
C_{dow}	NP-c*	NP-c*	NP-c*	P
C_k	P	NP-c	P	–

Table 1: NP-c* stands for NP-complete with the restriction on M, – means not applicable, X/Y means that the result for fuzzy CSPs (X) and the result for weighted CSPs (Y) are different.

References

Ahuja, R.; Magnanti, T.; and Orlin, J. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Arrow, K. J., and amd K. Suzumura, A. K. S. 2002. *Handbook of Social Choice and Welfare.* North-Holland.

Bacchus, F., and Grove, A. 1995. Graphical models for preference and utility. In *UAI*.

Bartholdi, J. J.; Tovey, C. A.; and Trick, M. A. 1992. How hard is it to control an election. In *Mathematical and Computer Modeling*, 27–40.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint solving and optimization. *Journal of the ACM* 44(2):201–236.

Bodlaender, H. L. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science* 209(1-2):1–45.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR* 21(1):135–191.

Brafman, R. I.; Rossi, F.; Salvagnin, D.; Venable, K. B.; and Walsh, T. 2010. Finding the next solution in constraint- and preference-based knowledge representation formalisms. In *Proc. KR 2010*.

Christian, R.; Fellows, M.; Rosamond, F.; and Slinko, A. 2007. On complexity of lobbying in multiple referenda. *Review of Economic Design* 11(3):217–224.

Dalla Pozza, G.; Pini, M. S.; Rossi, F.; and Venable, K. B. 2011. Multi-agent soft constraint aggregation via sequential voting. In *IJCAI*, 172–177.

Dalla Pozza, G.; Rossi, F.; and Venable, K. B. 2011. Multi-agent soft constraint aggregation:a sequential approach. In *ICAART*, 277–282.

Dechter, R. 2006. Tractable structures for CSPs. In Rossi, F.; Van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier.

Elkind, E.; Faliszewski, P.; and Slinko, A. 2009. Swap bribery. *Algorithmic Game Theory* 299–310.

Faliszewski, P.; Hemaspaandra, E.; Hemaspaandra, L. A.; and Rothe, J. 2007. Llull and copeland voting broadly resist bribery and control. In *Proceedings of AAAI'07*, 724–730.

Faliszewski, P.; Hemaspaandra, E.; and Hemaspaandra, L. 2009. How hard is bribery in elections? *JAIR* 35:485–532.

Faliszewski, P. 2008. Nonuniform bribery. In *Proc. AAMAS 2008*, 1569–1572.

Gibbard, A. 1973. Manipulation of voting schemes: A general result. *Econometrica* 41:587–601.

Kemeny, J. G. 1959. Mathematics without numbers. *Daedalus* 88:571–591.

Lang, J., and Xia, L. 2009. Sequential composition of voting rules in multi-issue domains. *Mathematical Social Sciences* 57(3):304–324.

Mattei, N.; Pini, M. S.; Venable, K. B.; and Rossi, F. 2012a. Bribery in voting over combinatorial domains is easy. In *Proceedings of ISAIM'12*.

Mattei, N.; Pini, M. S.; Venable, K. B.; and Rossi, F. 2012b. Bribery in voting over combinatorial domains is easy (extended abstract). In *Proceedings of AAMAS'12*, 1407–1408.

Mattei, N.; Pini, M. S.; Venable, K. B.; and Rossi, F. 2013. Bribery in voting with CP-nets. *Annals of Mathematics and Artificial Intelligence*.

Pini, M. S.; Venable, K. B.; and Rossi, F. 2013. Resistance to bribery when aggregating soft constraints (extended abstract). In *Proceedings of AAMAS'13*.

Rollon, E.; Flerova, N.; and Dechter, R. 2011. Inference schemes for m best solutions for soft CSPs. In *Proc. SOFT'11*.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Thorup, M. 1998. All structured programs have small tree width and good register allocation. *Information and Computation* 142(2):159–181.