Resistance to bribery when aggregating soft constraints, and exploitation of bribery cost schemes in preference compilation and optimization

Alberto Maran, Maria Silvia Pini, Francesca Rossi, and K. Brent Venable

Department of Mathematics, University of Padova, Italy
{amaran@studenti.math.unipd.it
{mpini,frossi,kvenable}@math.unipd.it

Abstract. We consider a multi-agent scenario, where the preferences of several agents are modelled via soft constraint problems and need to be aggregated to compute a single "socially optimal" solution. We study the resistance of various ways to compute such a solution to attempts to influence the result, such as those based on the notion of bribery. In doing this, we link the cost to bribe an agent to the effort needed by the agent to make a certain solution optimal, by only changing preferences associated to parts of the solution. This leads to the definition of four notions of distance from optimality of a solution in a soft constraint problem. The notions differ on the amount of information considered when evaluating the effort. We then show how to pass from such distance notions to suitable linearizations of the solution preference ordering, which can be exploited in the context of computing sets of k best solutions. We also show how the considered distances can be used in preference compilation tasks, such as when encoding elicited solution preferences in the constraint structure.

Keywords: bribery, soft constraint aggregation, preference optimization

1 Introduction

Often agents need to cooperate to take a collective decision. By doing this, the decision can be better than what they would have chosen, had they reasoned in isolation. Examples are collections of experts that have suggestions on what to do, which are then aggregated to obtain a single suggestion. Such experts could be, for example, classifiers in machine learning tasks, or web page rankers in web search.

We model such scenarios via a collection of agents that express their preferences over a common set of solutions to a problem. We assume that such preferences are described by soft constraints [15], more precisely either fuzzy and weighted constraints.

Agents' preferences are aggregated to compute a single "socially optimal" solution. To model this process, we consider some voting rules [2]. Although voting rules have been defined and studied in the context of political elections, they do exactly what we want: aggregating individual's preferences into a single collective "winner".

We then study the resistance of this setting, considering different voting rules, to external or internal attempts to influence the result. This happens often in political elections, but it could occur also in our settings. For example, when voting on a Doodle

event to choose a date for a meeting, if one participant sees how the others have voted (and thus can compute the result by considering these votes and her true vote), she could vote in a strategic way (that is, differently to what her true vote would say) in order to get a better result for her. This example is an instance of the so-called manipulation, where one or more agents may misreport their votes in order to get a better solution. Another kind of attempt may come from an external agent, usually called the "briber", who has a preferred solution, and tries to get that solution as the result of the voting process, by paying some agents to vote in a certain way, and by doing this while staying within its budget. In defining bribing scenarios, 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.

Classical results on voting theory tell us that every voting rule can be influenced by such attempts [2]. However, for some voting rules, it may be computationally difficult for the manipulators, or the briber, to understand how to design the attempt. Such rules are then said to be resistant to these attempts [3, 13].

In this paper we study whether our soft constraint aggregation scenarios are resistant to bribery. Resistance to manipulation has been studied already, for example in [7]. We consider two main approaches to aggregate the preferences: a sequential one, where agents vote on each variable at a time, and a one-step approach, where agents vote just once on entire solutions. We then define seven cost schemes to compute the cost of satisfying a briber's request. We find out that the one-step approach (which uses the Plurality voting rule) is not resistant to bribery. On the other hand, the sequential approaches (which are based on voting rules such as Plurality, Approval, and Borda), are all resistant to bribery for five out of seven cost schemes. This is very interesting, since the sequential approaches are also better in terms of complexity of determining the collective solution.

The cost schemes used in the bribery setting can be seen as a measure of the effort for an agent to respond to a briber's request. If the request is related to making a certain solution, say A, optimal (which means voting for it, if we use Plurality), then the cost can be considered a measure of how much the agent needs to change in its soft constraint problem in order to make A optimal. By following this line of reasoning, we exploit some of the cost schemes used for bribery to define four notions of distance from optimality of a solution in a soft constraint problem. We then show how to pass from such distance notions to corresponding linearizations of the solution preference ordering, which can be exploited in the context of computing sets of k best solutions. The computational complexity results obtained for the bribery problem can then be useful to determine how expensive it is to compute the top k solutions.

We also show how these distances can be used in preference compilation tasks, such as when encoding optimal solutions in the constraint structure. Making a solutions optimal according to a certain distance notion has the same computational complexity as determining the bribery cost with the corresponding cost scheme.

In the following, the formal proofs of some results have been omitted for lack of space.

2 Background

Soft constraints. A soft constraint [15] involves a set of variables and associates a value from a (partially ordered) set to each instantiation of its variables. Such a value is taken from a c-semiring, which is 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 respectively the worst and best element. 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), A is the set of preference values.

An instance of the SCSP framework is obtained by choosing a specific c-semiring. For instance, a classical CSP [9, 16] is just an SCSP where the c-semiring is $S_{CSP} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$. By choosing $S_{FCSP} = \langle[0, 1], max, min, 0, 1\rangle$ instead it means that preferences are in [0,1] and we want to maximize the minimum preference. This is the setting of fuzzy CSPs (FCSPs) [15], that we will use in the examples of this paper. In the paper we will also consider the setting of weighted CSPs (WCSPs), where the c-semiring is $S_{WCSP} = \langle R^+, min, +, +\infty, 0\rangle$, which means that preferences are interpreted as costs from 0 to $+\infty$, and that we want to minimize the sum of the costs.

Figure 1 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.

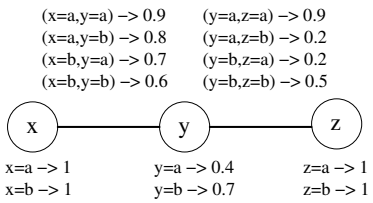

Fig. 1. A tree-shaped FCSP.

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 WSCSPs, 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). Given a variable x, we write $s \downarrow x$ to denote the value of x in s.

Given an FCSP Q and a preference α, we will denote as $cut_\alpha(Q)$ the CSP obtained from Q allowing only tuples with preference greater than or equal to α. It is known that the set of solutions of Q with preference greater than or equal to α coincides with the set of solutions of $cut_\alpha(Q)$.

Finding an optimal solution is an NP-hard problem, unless certain restrictions are imposed, such as a tree-shaped constraint graph. Constraint propagation may help the search for an optimal solution. Given a variable ordering o, a 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 instance of the SCSP approach by replacing max with $+$ and min with $\times$. Therefore, for WCSPs it is sufficient to replace max with min and min with sum.

DAC 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 [15]. In fact, such an optimum preference level is the best preference level in the domain of the root variable.

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 [2], given a set of candidates C, a *profile* is a collection of total orderings 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 "agents" the voters, "solutions" the candidates, and "decision" or "best solution" the winning candidate.

Some examples of widely used voting rules, that we will study in this paper, are:

- *Plurality*: each voter states a single preferred candidate, and the candidate who is preferred by the largest number of voters wins;
- *Borda*: 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;
- *Approval*: given m candidates, each voter approves between 1 and $m-1$ candidates, and the candidate with most votes of approval wins.

We know that every voting rule is manipulable [2]. However, if it is computationally difficult to influence the result by using a certain voting rule, we can say that the voting rule is *resistant* to such attempts. Thus the computational complexity of various attempts to influence the result of the voting process has been studied [3, 13, 6]. Besides manipulation, which refers to scenarios where there is a voter (or a group of voters) who can get a better result by lying on its preference ordering, another kind of attempt to influence the result is called *bribery*: 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.

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, this means that 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 [8] this is the notion of *soft profile*, which is formally defined as 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[1]. 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 Fig. 2.

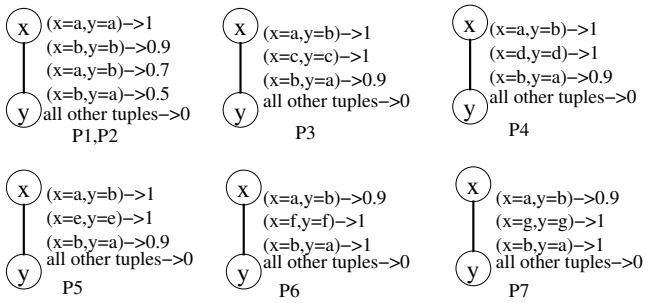

Fig. 2. A fuzzy profile.

The idea proposed in [8, 7] 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 such variables $O = \langle v_1, \ldots, v_n \rangle$, and a corresponding sequence of voting rules $R = \langle r_1, \ldots, r_n \rangle$ (that will be called "local rules"). The sequential procedure is a sequence of n steps, where at each step i,

- All agents are first 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 applied to propagate its effect considering the reverse ordering of O.

[1] Notice that 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.

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 $Seq_{O,R}(V, D, P)$. In the soft profile above, assume the variable ordering is $\langle x, y \rangle$ and $r_i =$ Approval for all $i = 1, 2$. In step 1, agents apply DAC. This changes the preferences of the agents over x. For example, in P_1 and P_2, $x = a$ maintains preference 1, $x = b$ gets preferences 0.9, and all other domain values get preference 0, while in P_3, $x = a$ and $x = c$ maintain preference 1, $x = b$ gets preference 0.9, while all other values get preference 0. Then, Approval is applied on the profile over x where the sets of approved values are: $\{a\}$ for the first two voters and respectively $\{c, a\}$, $\{d, a\}$, $\{e, a\}$, $\{f, b\}$, and $\{g, b\}$ for the others. Thus, $x = a$ is chosen and the constraint $x = a$ is added to all SCSPs, and its effect is propagated via DAC on the domain of y. In step 2, DAC does not modify any preference (since y is the last variable) and the sets of approved values for y are the same for all agents and contain only b. Thus the elected solution with the sequential procedure is $s = (x = a, y = b)$, which has preference 0.7 for P_1 and P_2, 1 for P_3, P_4, and P_5, and 0.9 for P_6 and P_7.

An alternative to this sequential procedure would be to generate the preference orderings for each voter from their FCSPs, and then to aggregate them in one step via a voting rule, for example Approval. In our example, $(x = a, y = b)$ gets 3 votes (that is, it is optimal for 3 agents), $(x = a, y = a)$ and $(x = b, y = a)$ each gets 2 votes, $(x = f, y = f)$, $(x = d, y = d)$, $(x = e, y = c)$, $(x = c, y = e)$, and $(x = g, y = g)$ each gets 1 vote, while all other solutions get no vote. Thus the winner is $(x = a, y = b)$.

Kemeny distances. The Kemeny distance between two orderings is the number of pairs on which they differ [14]. 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 $\sum_{i,j=1,\, i<j}^{n} |(sgn(a_i - a_j) - sgn(b_i - b_j)|$. For example, given $\Omega = \{x_1, x_2, x_3, x_4\}$, the Kemeny distance between $o_1 = (1, 2, 3, 4)$ (i.e., $x_1 > x_2 > x_3 > x_4$) and $o_2 = (1, 2, 4, 3)$ (i.e., $o_2 : x_1 > x_2 > x_4 > x_3$) is $k(o_1, o_2) = 2$, since o_1 and o_2 rank in a different order only the pairs (x_3, x_4) and (x_4, x_3).

The weighted Kemeny distance allows to discriminate situations in which the differences between two orderings are in the top pairs rather than in the bottom ones, since the former differences may be considered more serious. Given a set of candidates Ω, two orderings o_1 and o_2 over Ω, a weighting vector $w = (w_1, \ldots, w_{n-1}) \in [0, 1]^{n-1}$ such that $w_1 \geq \ldots \geq w_{n-1}$ and $\sum_{i=1}^{n-1} w_i = 1$, the *weighted Kemeny distance* between o_1 and o_2, say $kw(o_1, o_2)$, is $\frac{1}{2}[\sum_{i,j=1,\, i<j}^{n} w_i |(sgn(a_i^{\sigma_1} - a_j^{\sigma_1}) - sgn(b_i^{\sigma_1} - b_j^{\sigma_1})| + \sum_{i,j=1,\, i<j}^{n} w_i |(sgn(b_i^{\sigma_2} - b_j^{\sigma_2}) - sgn(a_i^{\sigma_2} - a_j^{\sigma_2})|]$, where σ_1 and σ_2 are two permutations on $\{1, \ldots, n\}$ that correspond to the orderings o_1 and o_2. For example, if $w = (\frac{3}{6}, \frac{2}{6}, \frac{1}{6})$, the weighted Kemeny distance between the orderings o_1 and o_2 in the example above is $kw(o_1, o_2) = \frac{1}{3}$.

Given two orderings, it is polynomial to compute both their Kemeny distance and their weighted Kemeny distance.

3 The bribery problem

We consider scenarios where a collection of agents need to take a decision, by selecting it out of a set of possible decisions. Such decisions are described by the Cartesian product of the domains of a set of variables. These variables are shared by all agents. Each agent has its own preferences over such decisions, described via a set of soft constraints. In this paper, by soft constraints we mean either fuzzy or weighted constraints. Also, we assume that all agents have tree-shaped soft constraints problems.

Notice that the set of solutions of such constraint problems (that is, the set of decision among which to choose one) is in general exponentially large w.r.t. the size of the soft constraint problems. We also assume that the number of such solutions is exponentially large w.r.t. the number of agents.

We now formally define the bribery problem of which we will study the computational complexity:

Definition 1. *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 according to cost scheme C and by spending less than a certain budget .*

3.1 Winner determination

It makes sense to consider only winner determination approaches which are polynomial to compute: if it is difficult to compute the winning decision, it is also difficult for a briber to compute how to bribe the agents (since he needs to know who the winner is without the bribery).

We consider two main approaches: sequential and one-step. For the sequential approach, we employ the sequential voting procedure described in the previous section. We have an ordering O over the variables, and we are going to 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:

- Sequential Plurality (SP): one best value for X;
- Sequential Approval (SA): all best values for X;
- 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 follow:

- SP and SA: the value voted by the highest number of agents;
- SB: the value with best score, where the score of a value is the sum of its preferences over all the agents; notice that "best" here means maximal in the case of fuzzy constraints, while it is the minimal in the case of weighted constraints.

Once a value is chosen for a variable, this value is broadcasted to all agents, who fix variable X to this value in their soft constraints and apply DAC in the reverse ordering

w.r.t. O. We then 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. In this case, a possible voting rule to use is what we call One-step Plurality (OP), where each agent provides an optimal solution of his soft constraint problem, and then we select the solution which is provided by the highest number of agents.

For all the voting rules we consider (SP, SA, SB, and OP), it is computationally easy for an agent to vote. An approach like OP is however less satisfactory that the sequential approaches in terms of *ballot expressiveness*: since the number of solutions is 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 to the number of solutions they can vote for, and this means that in all cases an exponentially large number of solutions will not be voted. So there is trade-off between easiness of computing votes and ballot expressiveness.

3.2 Bribery actions and cost schemes

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.

To define the cost of a briber's request, which is to make a certain solution A optimal, we consider the following approaches:

- 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 of A, denoted with $pref(A)$, to the optimal preference of the soft constraint problem of the agent, denoted with opt. If we are dealing with fuzzy numbers and we may prefer to have integer costs, the cost will be defined as $C_{do} = (opt - pref(A)) * l$, where l is the number of different preference values allowed. With weighted constraints, if costs are natural numbers, we may define $C_{do} = pref(A) - opt$, since opt is the smallest cost.
- C_{don}: The cost is determined by considering both the distance between the preference of A and the optimal preference, and the number of tuples, say t, of A that must be modified in order to make A optimal. Thus, if we have n variables, with fuzzy constraints we may define $C_{don} = ((opt - pref(A)) * l * M) + t$, where M is a large integer and $1 \leq t \leq n$. If instead we consider weighted constraints, we define $C_{don} = ((pref(A) - opt) * M) + t$. In both case, the role of M is to assure a higher bribery cost for a less preferred candidate: we want that the highest cost at a given preference level for A, that is, $d * M + 2m - 1$, where $d = (opt - pref(A)) * l$ and m is the number of variables, to be smaller than the lowest cost at the next preference level, that is, $(d + 1)M + 1$. This yields $M > 2m - 2$.

- C_{dow}: The cost is computed by considering the same as in C_{don}, but each preference to be modified is associated with a cost proportional to the change required on that preference. If we denote by t_i any tuple of A with preference $\leq opt$, then the cost will be $((opt - pref(A)) * l * M) + \sum_{t_i}(opt - pref(t_i)) * l$ for fuzzy constraints, where the role of M is similar to the one in C_{don}. For weighted constraints, we analogously define $C_{dow} = ((pref(A) - opt) * M) + \sum_{t_i}(pref(t_i) - opt)$. However, it is easy to see that $\sum_{t_i}(pref(t_i) - opt) = pref(A) - opt$, thus we have $C_{dow} = ((pref(A) - opt) * (M + 1))$.
- C_{donw}: The cost is the combination of C_{don} and C_{dow}. For fuzzy constraints: $C_{donw} = ((opt - pref(A)) * l * M) + t * M' + \sum_{t_i}(opt - pref(t_i)) * l$, where M' has a similar role than M w.r.t. the second a third component of the sum. For weighted constraints: $C_{donw} = ((pref(A) - opt) * M) + t$ (by simplifying as in C_{dow}).

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. Notice that in these cases the briber's request can be to make a certain value optimal (for SP), or to make a certain set of values optimal (for SA), or to vote with a certain ordering (for SB). We thus consider the following cost schemes:

- C_k: The cost is 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.
- C_{kw}: The cost is the same as for C_k, but considering the weighted Kemeny distance.

4 Winner and cost determination are both computationally easy

As noted above, voting is easy. It is easy to check that also computing the winner is easy with any of the voting rules we consider. Thus we have the following result:

Theorem 1. *Winner determination takes polynomial time for SP, SA, SB, and OP.*

Proof: For each variable, SP (resp., SA) requires a (resp., all) the most preferred values in the domain of that variable. SB instead requires an ordering over such values. 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 applying 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. □

It is polynomial also to compute the cost to respond to a briber's request, for all the cost schemes we have defined.

Theorem 2. *Given a tree-shaped fuzzy or weighted CSP and an outcome A, determining the cost to make A an optimal outcome takes polynomial time for C_{equal}, C_{do}, C_{don}, C_{dow}, and C_{donw} .*

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}, C_{dow}, and C_{donw}, we need to compute opt, the numbers of tuples of A with preference worse than opt, and the distance of their preferences from opt. All of these components can be computed in polynomial time. with tree-shaped problems. □

Theorem 3. *Given a tree-shaped fuzzy or weighted CSP, one 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 both C_k and C_{kw}, regardless of the choice of the weights.*

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 in the domain of X, it is enough to modify their preferences to make them optimal in the domain of X. We can then compute the Kemeny (weighted) distance between the old and the new ordering, which is polynomial since the domain of X has bounded cardinality. □

5 Resistance to bribery

We now study the resistance to bribery of SP, SA, and SB.

5.1 Voting sequentially

Theorem 4. *(V, C)-Bribery is NP-complete (and also W[2]-complete) for $V \in \{SP, SA, SB\}$ and $C \in \{C_{equal}, C_{do}\}$.*

Proof: Membership in NP is easy to prove. To show completeness, we provide a polynomial reduction from the OPTIMAL LOBBYING (OL) problem [5]: we are given an $n \times m$ $0/1$ matrix E and a $0/1$ vector $\boldsymbol{x}$ of length m where each column of E represents an issue and each row of E represents a voter. E is a binary matrix with 1 corresponding to a "yes" vote and $\boldsymbol{x}$ 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 $\boldsymbol{x}$. This problem is shown to be $W[2]$-complete. By giving a polynomial reduction from OL to our bribery problems, we show that our problem is NP-complete (actually $W[2]$-hard with parameter the budget B).

Given an instance $(E, \boldsymbol{x}, k)$ of OL, we construct an instance of (V,C_{do})-Bribery, where $V \in \{SP, SA, SB\}$, containing soft constraints with only binary variables and no constraints. The number of variables, m, is 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: if we use fuzzy constraints, for each variable the value indicated in the row will have preference 1, while the other value will have preference 0; with weighted constraints, 1 and 0 will become costs 0 and $+\infty$.

Each voter has a unique most preferred solution with preference 1 (or cost 0) and all other solutions have preference 0 (or cost $+\infty$). We set the preferred outcome $A = \boldsymbol{x}$.

This means that according to C_{do}, all voters not voting for A have the same cost to be bribed, which is $(opt - pref(A)) * 100 = (1-0) * 100 = 100$ assuming 100 preference levels are allowed, resp., $pref(A) - opt) = +\infty - 0 = +\infty$. Finally, we set the budget $B = k * 100$, resp., $B = k$. With C_{equal}, the cost is always 1 if A is not already voted for. 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 $\boldsymbol{x}$ becomes the winning agenda of the OL instance. □

Theorem 5. *(V,C)-Bribery is NP-complete (and also W[2]-complete) for $V \in \{SP, SA, SB\}$ and $C \in \{C_{don}, C_{dow}, C_{donw}\}$, if $M > n * m$, where n is the number of voters and m the number of variables.*

Proof: Membership in NP is easy to prove. To show completeness, we provide a polynomial reduction from the OPTIMAL LOBBYING (OL) problem [5]. In this problem, we are given an $n \times m$ $0/1$ matrix E and a $0/1$ vector $\boldsymbol{x}$ 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 $\boldsymbol{x}$ 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 $\boldsymbol{x}$. This problem is shown to be $W[2]$-complete. By giving a polynomial reduction from OL to our bribery problems, we show that our problem is NP-complete (actually $W[2]$-hard with parameter the budget B). Given an instance $(E, \boldsymbol{x}, k)$ of OL, we construct an instance of (V-C_{do})-Bribery, where $V \in \{SP, SA, SB\}$, containing soft constraints with only independent binary variables. The number of variables, 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. In particular, if we use fuzzy constraints, 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; with weighted constraints, the value indicated in the row will be associated with cost 0 while the other value will be associated with cost $+\infty$. Thus, each voter has a unique most preferred solution with preference 1 (or cost 0) and all other complete assignments have preference 0 (or cost $+\infty$). We set the preferred outcome $A = \boldsymbol{x}$. This means that according to C_{do}, all voters not voting for A have the same cost to be bribed, which is $(opt - pref(A)) * 100 = (1 - 0) * 100 = 100$ assuming 100 preference levels are allowed, resp., $pref(A) - opt) = +\infty - 0 = +\infty$. Finally, we set the budget $B = k * 100$, resp., $B = k$. 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 $\boldsymbol{x}$ becomes the winning agenda of the OL instance. □

Theorem 6. *(SA,C)-Bribery is NP-complete for $C \in \{C_k, C_{kw}\}$.*

Proof: Follows from NP-completeness of bribery for Approval in the single variable case [12, 11]. □

Theorem 7. *(SP,C)-Bribery and (SB,C)-Bribery are in P for $C \in \{C_k, C_{kw}\}$, assuming ties are broken in favor of the briber.*

Proof: 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. Plurality has been shown to be in P in [12] even with non-uniform costs. This, together with the fact that computing the cost is polynomial for both C_k and C_{kw} (Theorem 3), and that ties are broken in favor of the briber, allows us to conclude for SP. A similar proof works for Borda. □

5.2 Voting with OP

We now show that OP is not resistant to bribery. To do this, we will need to compute n cheapest alternative candidates for each agent to vote for. We will thus start by studying the computational complexity of this task.

Theorem 8. *Given a tree-shaped fuzzy or weighted CSP, computing a set of k cheapest outcomes according to C_{do} and C_{equal} is in $\mathcal{P}$.*

Proof: The cost of a solution according to C_{do} is an integer proportional to the distance between the preference of the solution and the preference of an optimal solution. 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 [4] 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 [10].

With C_{equal}, the cost is 0 for all the optimal solutions and 1 for the other ones. Thus any of the above procedures can be used to compute the top k solutions in polynomial time. □

Theorem 9. *Given a tree-shaped weighted CSP, computing a set of k cheapest outcomes according to C_{dow} is in $\mathcal{P}$.*

Proof: Follows from the fact that, for weighted CSPs, C_{dow} is proportional to C_{do}. □

For the other cost schemes, we define a general algorithm, called $KCheapest$, that will work for C_{don}, as well as C_{dow} and C_{donw}, via small modifications. In what follows we assume a voter represents his preferences with a tree-shaped fuzzy CSP. The input to $KCheapest$ is a tree-shaped fuzzy CSP P, an integer k, and a cost scheme C. The output is a set of k cheapest solutions of P according to C.

$KCheapest$ performs the following steps:

1. **Find k optimal solutions of P, or all optimal solutions if they are less than k.** If the number of solutions found is k, we stop, otherwise let k' be the number of remaining solutions to be found.

2. **Look for the remaining top solutions within non-optimal solutions**. More in detail, until k' best solutions have been found or all solutions of P have been exhausted, consider each preference pl associated to some tuple in P in decreasing order and, for each tuple t of P with preference pl, perform the following:
   (a) Compute the new fuzzy CSP, P_t, obtained by fixing the tuple in the constraint (that is, by forbidding all other tuples in that constraint).
   (b) Compute a new soft CSP, say P_t^w, associated to P_t, defined as follows:
      i. the constraint topology of P_t^w and P_t coincide;
      ii. each tuple with a preference greater or equal than opt in P_t has weight 0 in P_t^w;
      iii. each tuple with a preference pt such that $pl \leq pt < opt$ in P_t has weight c in P_t^w defined as follows: $c = 1$ if $C = C_{do}$, $c = pt - opt$ if $C = C_{dow}$ and $c = (1, pt - opt)$ if $C = C_{donw}$;
      iv. each tuple with preference less than pl in P_t has weight $+\infty$ in P_t^w.

      Thus, P_t^w is a weighted CSP if $C = C_{don}$ or $C = C_{dow}$, while it is a SCSP defined on the Cartesian product of two weighted semirings if $C = C_{donw}$.
   (c) Compute the k' best solutions of all the solutions if they are less than k' of P_t^w.

   Take the k' top solutions (or all solutions if less than k') among the sets of best solutions computed for P_t^w, $\forall t$ such that $pref(p) = pl$.

Theorem 10. *Given a tree-shaped fuzzy CSP P, computing a set of k cheapest outcomes according to C_{don}, C_{dow}, and C_{donw} is in $\mathcal{P}$.*

Proof: We prove that the solutions returned by algorithm $KCheapest$ are indeed the k cheapest (or all the solutions if the k exceeds the total number of solutions) according to the selected cost scheme (depending on how the weights are defined in step (iii)) and that $KCheapest$ runs in polynomial time. We assume that k is smaller than the cardinality of the set of all solutions, but a similar reasoning holds for the other case. For all cost schemes, optimal solutions are always cheaper than other solutions. Thus step 1 is correct. In step 2, the solutions of any P_t^w correspond to solutions of P with preference pl, and considering all such problems allows to cover all solutions of P with such a preference. The way weights are defined in P_t^w allows to order solutions with the same preference, respectively, in increasing order either w.r.t. the number of tuples that need to be changed in order to make the solution optimal ($c = 1$), or w.r.t. the weighted sum of the changes ($c = opt - pt$) needed to make the solutions optimal, or in lexicographic order with respect to these two criteria where the number of tuples to be changed comes first ($c = (1, opt - pt)$). These three ways of breaking ties among solutions with the same preference correspond, respectively, to C_{don}, C_{dow}, and C_{donw}. All remaining ties are assumed to be broken using a lexicographic ordering induced by linear orders over the variables and the values in the domains.

In terms of computational complexity, step 1 is achieved by computing the optimal preference level opt, and obtaining the tree-shaped CSP corresponding to the opt-cut of P, denoted with P^{opt}. Then $KCheapest$ finds a set of k solutions of P^{opt}. This is done by exploiting a lexicographic order over the solutions, by finding the first optimal solution according to such an order, and by iteratively finding the following next best solutions, until either $k - 1$ steps have been performed, or the set of solutions has

been exhausted. This can be done in polynomial time as shown in [4]. Also step 2 is polynomial, since computing the k best solutions on a weighted tree-shaped CSP can be done in polynomial time by using the procedure in [10]. □

Theorem 11. *(OP, C)-Bribery is in $\mathcal{P}$ for $C \in \{C_{equal}, C_{do}, C_{don}, C_{dow}, C_{donw}\}$ 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 $\mathcal{P}$. To solve the decision problem for a certain r, we transform this problem to a minimum-cost flow problem [1]. The network has a source s, a sink t, and three "layers" of nodes.

The first layer 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, for a total of at most $2n + 1$ candidates. 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 other non-voted candidates that we do not include in the second layer would never be used in the new profile obtained by bribing, since they cost too much. Providing n non-voted candidates for each voter is enough, since there are n voters and in the worst case each of them votes for a different candidate. For each node 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 to bribe 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 Section **??**. In general, it is sufficient to compute the $2n$-best 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 given the results in Section 4.

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 every candidate besides A cannot 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 [12], 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 minimal flow in the larger network which goes through one of the nodes which we omit. This means that a voter has been forced to vote for another, more expensive, non-voted candidate since all 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 edges. Since min cost feasible flow problem can be solved in polynomial time in the number of nodes and edges using for example the Edmond-Karp algorithm [1], the overall running time of this method is polynomial. □

5.3 Summary of bribery results

Our complexity results about the resistance to bribery when aggregating the preferences of a collection of agents, if they are modelled via soft constraints, can be seen in Table 1. We can see that OP is not resistant to bribery, since it is computationally easy for the briber to compute who to bribe and what to ask for, and to check whether he can do it within its budget. On the other hand, the sequential approaches (SP, SA, and SB) are all resistant to bribery, if agents compute costs according to C_{equal}, C_{do}, C_{don}, C_{dow}, or C_{donw}. SA is resistant to bribery also with C_k and C_{kw}. Thus, it is clear that sequential approaches should be preferred if resistance to bribery is an important feature.

Notice that, when a problem is polynomial for soft constraints, it is also so for CSPs. Thus, OP is easy to bribe also when agents use CSPs. The same is for SP and SB with the Kemeny-based costs.

	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_{donw}	NP-c*	NP-c*	NP-c*	P/?
C_k	P	NP-c	P	–
C_{kw}	P	NP-c	P	–

Table 1. Bribery complexity results. NP-c* stands for NP-complete with the restriction on M (and M' if present), – stands for not applicable. When the complexity results for fuzzy constraints and weighted constraints are different, we write X/Y, where X is the complexity for fuzzy and Y is the complexity for weighted constraints.

6 Bribery cost schemes in preference optimization and compilation

There are many constraint reasoning scenarios in which it is useful to consider, given a solution, how far it is from being optimal. One natural way to evaluate this is to consider the difference between its preference and the preference of an optimal solution, or one may want to take into account also the effort that would be required in terms of changes needed in the soft constraints to make such a solution optimal. This is the same kind of reasoning that led us to the bribery cost schemes defined in Section 3.2. Given a tree-shaped SCSP P, let us define for each solution s the following triple of values (p_s, t_s, w_s) where $p_s = opt(P) - pref(s)$, t_s is the minimum number of tuples of s that must be changed to make s optimal, and w_s is the sum of the amount of changes that must be performed on such tuples to make s optimal. Given such a triple, we can define the following notions of distance of a solution s from optimality:

- do: the distance is p_s, that is, the distance is computed only looking at the first component of the triple;
- don: the distance is determined by considering first p_s and then t_s;
- dow: the distance is determined by considering first p_s and then w_s;
- $donw$: the distance is determined by first considering p_s, then t_s, and then w_s.

Often finding just one optimal solution may not be enough and it may be desirable to produce a set of top solutions. This occurs, for example, in web search or configuration problems, where we usually want more than one answer to our query.

With soft constraints, usually the number of different preference values is much smaller than the number of solutions, thus many solutions end up having the same preference. When computing the k best solutions, it is thus necessary to employ a tie-breaking rule among solutions with the same preference value. Some of the notions of distance from optimality defined above provide a meaningful way to tie-break: in addition to the distance of the solution preference from the optimal preference, they consider also the "structural" distance of the solution from being optimal.

For example, among the solutions with the same preference, don will put first the ones that require the minimum number of tuples to be changed. Besides this, dow weights each tuple to be modified with the amount by which it must be modified. A refinement of don is $donw$, which considers the amount of changes needed only in case of a tie on both the preference distance and the number of tuples to be modified.

From results in [4] and [10], we know that computing the k best solutions according to do is polynomial, for both fuzzy and weighted constraints. The algorithms defined above to compute the cheapest top solutions allow us to give the following result on the complexity of computing the k best solutions according to the new refinements of the solution preference ordering.

Theorem 12. *Computing k best solutions is in $\mathcal{P}$ according to distances don, dow and $donw$ for tree-shaped fuzzy CSPs and according to dow for tree-shaped weighted CSPs.*

In the process of modeling a real-life problem via soft constraints, it can be reasonable to allow users to require that a certain solution be made optimal in the current soft constraint problem. To measure the effort needed to achieve this, we can use the distance notions defined above. Theorem 2 allows us to state the following result:

Theorem 13. *Computing the effort needed to make a solution optimal is in P when using fuzzy or weighted three-shaped constraint problems and any of the distance notions do, don, dow, and donw.*

7 Conclusions

We have studied resistance to bribery when aggregating preferences of several agents expressed via soft constraints. This has led to several results that are interesting and useful in themselves. However, they also have a wider applicability within typical CP tasks, such as computing the top k solutions and encoding solution preferences.

We believe that this is just a first step in a very promising multi-disciplinary research line, which creates useful links between CP issues and voting theory.

References

1. R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
2. K. J. Arrow and A. K. Sen amd K. Suzumura. *Handbook of Social Choice and Welfare.* North-Holland, 2002.
3. J.J. Bartholdi, C.A. Tovey, and M.A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.
4. R. I. Brafman, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. Finding the next solution in constraint- and preference-based knowledge representation formalisms. In *Proc. KR 2010*, 2010.
5. R. Christian, M. Fellows, F. Rosamond, and A. Slinko. On complexity of lobbying in multiple referenda. *Review of Economic Design*, 11(3):217–224, 2007.
6. V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *JACM*, 54(3):1–33, 2007.
7. G. Dalla Pozza, M. S. Pini, F. Rossi, and K. B. Venable. Multi-agent soft constraint aggregation via sequential voting. In *IJCAI*, pages 172–177, 2011.
8. G. Dalla Pozza, F. Rossi, and K. B. Venable. Multi-agent soft constraint aggregation:a sequential approach. In *ICAART*, pages 277–282, 2011.
9. Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
10. N. Flerova E. Rollon and R. Dechter. Inference schemes for m best solutions for soft CSPs. In *Proc. SOFT'11*, 2011.
11. E. Elkind, P. Faliszewski, and A. Slinko. Swap bribery. *Algorithmic Game Theory*, pages 299–310, 2009.
12. P. Faliszewski. Nonuniform bribery. In *Proc. AAMAS 2008*, pages 1569–1572, 2008.
13. P. Faliszewski, E. Hemaspaandra, and L.A. Hemaspaandra. How hard is bribery in elections? *JAIR*, 35:485–532, 2009.
14. J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.
15. P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In P. Van Beek F. Rossi and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2005.
16. F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.