An Integer Linear Programming Approach for Coalitional Weighted Manipulation under Scoring Rules

Antonia Maria Masucci, Alonso Silva

An Integer Linear Programming Approach for Coalitional Weighted Manipulation under Scoring Rules

Antonia Maria Masucci
Orange Labs
38-40 rue du Général Leclerc
92130 Issy-Les-Moulineaux
France
Email: antonia.masucci@orange.com

Alonso Silva
Bell Labs, Alcatel-Lucent
Centre de Villarceaux
Route de Villejust
91620 Nozay
France
Email: alonso.silva@alcatel-lucent.com

***Abstract*—In this work, we are interested to prove that for voting systems which can be expressed as scoring rules, the coalitional weighted manipulation problem which is known to be NP-complete is as difficult as solving an integer linear programming problem. For this integer linear programming problem several software solutions exist, and we have found that with a reasonable number of candidates the solution can be found within seconds.**

I. Introduction

Voting systems, also known as electoral systems, have gained increasing interest in recent years. In particular, the computational aspects of voting have been object of numerous researches. Voting systems establish rules to determine the value of votes and how to aggregate them.

The famous Gibbard-Satterthwaite theorem states that for any voting rule that is not a dictatorship, there are elections in which at least one of the voters would benefit by lying [1], [2]. A dictatorship is a voting rule where one of the voters (the dictator) single-handedly decides the outcome of the election.

The Gibbard-Satterthwaite Theorem implies that in theory, voters are able to manipulate elections. In practice, deciding which manipulation to employ may prove to be a hard computational problem. Indeed, there are a superpolynomial number of possibilities of ranking the candidates. Bartholdi et al. [6] showed a voting rule where manipulation is NP-hard. Conitzer et al. [7] showed that in a variety of voting rules, coalitional manipulation is NP-hard, even if there are only a constant number of candidates. A shortcoming of the result is that they are worst-case hardness results, and thus provide a poor obstacle against potential manipulators.

In this work, we analyze the computational aspects of manipulation. We propose an integer linear programming appoach and we realize that although the problem is NP-hard, in practice, with a small number of candidates the resolution of the manipulation problem is feasible and rather simple.

II. Preliminaries

An election $(\mathcal{V},\mathcal{C})$ consists of a set $\mathcal{V}=\{v_1,v_2,\ldots,v_N\}$ of voters and a set $\mathcal{C}=\{c_1,c_2,\ldots,c_M\}$ of candidates. Each voter submits a ranking of the candidates (which is a total order over the set $\mathcal{C}$).

The voting setting also includes a voting rule, which is a function from the set of all possible combinations of votes to the set of candidates. We shall discuss the voting rules related to scoring rules. Whenever the voting rule is based on scores, the candidate with the highest score wins.

Examples of voting rules based on scores are [3]:

- *Plurality*: Each voter assigns a point to the preferred candidate and the candidate who receives the largest score wins;
- *Veto*: Each voter "vetoes" a candidate and the candidate that is vetoed by the fewest voters wins;
- *Borda*: Given m candidates, each voter gives a ranking of all candidates, the j-th ranked candidate score is $m-j$, and the candidate with largest sum of score wins.

Let the scores $x=(x_1,x_2,\ldots,x_M)$ be a vector of non-negative integers such that $x_1\geq x_2\geq\ldots\geq x_M$. For each voter, a candidate receives x_1 points if it is ranked first by the voter, x_2 if it is ranked second, etc. The score s_m of a candidate is the total number of points the candidate receives.

For example, for Plurality scoring rule we have $x=(1,0,\ldots,0)$, for Veto scoring rule we have $x=(1,1,\ldots,1,0)$, for Borda scoring rule we have $x=(m-1,m-2,\ldots,0)$, etc.

In the constructive coalitional weighted manipulation (CCWM) problem [4], we are given a set $\mathcal{C}$ of candidates, with a distinguished candidate $p\in\mathcal{C}$, a set of weighted voters S that already cast their votes (those are the truthful voters), and weights for a set of voters T that still have not cast their votes (the manipulators). We are asked whether there is a way to cast the votes in T such that p wins the election. The constructive coalitional unweighted manipulation (CCUM) problem is a special case of CCWM problem where all the weights equal 1 [5].

III. Main results

A. Unweighted manipulation

Consider M candidates and N voters. Every vote can be seen as a permutation of the vector of values

$$x = (x_1, \dots, x_M)^T$$

where $x_m \geq 0$, for all $m \in \{1, \dots, M\}$ where we consider

$$x_1 \geq x_2 \geq \ldots \geq x_M$$

and through translation and scaling without loss of generality we can assume that $x_1 = 1$ and $x_M = 0$ so that

$$1 = x_1 \geq x_2 \geq \ldots \geq x_M = 0.$$

We consider

$$\sigma^n(x) = (\sigma^n(x_1), \sigma^n(x_2), \dots, \sigma^n(x_M))$$

the permutation of x which gives the vote of voter n, for $n \in \{1, \dots, N\}$.

The final score for a candidate $m \in \{1, \dots, M\}$, denoted by s_m, can then be written as

$$s_m = \sum_{n=1}^{N} \sigma^n(x_m)$$

The total final score is denoted as

$$s = (s_1, s_2, \dots, s_M).$$

Since we are considering the constructive coalitional weighted manipulation (CCWM) problem, we assume that we know $S \in \mathbb{N} \cup \{0\}$ votes and without loss of generality we consider that they are the votes of the first voters, i.e., $\sigma^1(x), \sigma^2(x), \dots, \sigma^S(x)$. We can then choose the remaining $N - S$ votes $\sigma^{S+1}(x), \dots, \sigma^N(x)$.

We denote by u the combined votes of the S known votes, i.e.,

$$u = \left(\sum_{i=1}^{S} \sigma^i(x_m) \right)_{m \in \{1,2,\dots,M\}}$$

Without loss of generality, we assume that our objective is to elect the first candidate, i.e., the distinguished candidate p corresponds to the first candidate c_1. Thus, the first coordinate of the votes we can choose are given by

$$\sigma^i(x_1) = 1 \quad \forall S + 1 \leq i \leq N.$$

Since we are dealing with a scoring rule, it is always better to choose the maximum score to the distinguished candidate.

Therefore,

$$s_1 = u_1 + (N - S).$$

To see the differential, denoted by $\hat{x}$, between the first candidate and all the other candidates of the known votes, we consider

$$\hat{x} = u - s_1 \mathbb{1},$$

where $\mathbb{1} = (1, 1, \dots, 1)$ is the $m \times 1$ vector.

We notice that $\hat{x}_1 = 0$. Our objective is that $\hat{x}_k < 0$ for all $k \in \{2, \dots, M\}$. Equivalently, our objective is that there exists $\varepsilon > 0$ such that $\hat{x}_k < -\varepsilon$ for all $k \in \{2, \dots, M\}$.

There are $M!$ possible permutations of x. If M is a small number, then we can enumerate the possible permutations in a matrix $\hat{A}$. We are thus looking for a vector $\alpha = (\alpha_1, \dots, \alpha_{M!})$ such that $\hat{A}\alpha \leq 0$ subject to the constraint that $\alpha \cdot \mathbb{1} = N - S$.

We assume that we want more than the fact that ε is positive, we want to maximize ε such that $\hat{A}\alpha < -\varepsilon\mathbb{1}$ in order to maximize the differential with the second closest candidate. or equivalently, by defining $A = -\hat{A}$ the constraint becomes $A\alpha > \epsilon$

This is an integer linear programming given by

$$\begin{aligned} &\max \varepsilon \\ &\quad A\alpha > \epsilon \\ &\alpha \cdot \mathbb{1} = N - S \\ &\quad \alpha \in \mathbb{N} \cup \{0\}. \end{aligned}$$

B. Weighted manipulation

A weight function is a mapping $w : \mathcal{V} \to \mathbb{N}$. When voters are weighted, the above rules are applied by considering a voter of weight ℓ to be ℓ different voters. The method to convert the weighter manipulation problem to an integer linear programming is similar to the previous method and thus we do not repeat it here.

IV. Simulations

In this section, we show an example of our integer linear programming approach applied to a voting system, based on Borda voting rules, with four candidates. Then, we generalize generalize the previous approach to any number of candidates and we present the proposed algorithm.

A. Four candidates

Consider that we have 4 candidates, denoted A, B, C and D. The voting rule is Borda, which means that for each vote, the favorite candidate will receive a score of 3, the second favorite candidate will receive a score of 2, the third favorite candidate, a score of 1, and the last favorite candidate will receive a score of 0. Through the transformation of translation and scaling we obtain that it is equivalent to the fact that the scores are given by $(1, 2/3, 1/3, 0)$.

Let us suppose that the total number of votes is 120000 and that the manipulator votes are 25% of those, i.e., 30000 votes.

Consider that we know the remaining 90000 truthful votes:

- 15000 are $B > C > A > D$
- 15000 are $B > D > C > A$
- 30000 are $C > D > A > B$
- 30000 are $D > A > B > C$

And consider that we want to make the first candidate to win the election, i.e., candidate A. Following the algorithm

proposed in the previous section, the first step is to compute the current scores of each candidate.

It is easy to see that the scores are given as follows:

$$A = 30000 * (2/3) + 45000 * (1/3) = 35000$$
$$B = 30000 * (1) + 30000 * (1/3) = 40000$$
$$C = 30000 * (1) + 15000 * (2/3) + 15000 * (1/3) = 45000$$
$$D = 30000 * (1) + 45000 * (2/3) = 60000.$$

According to the previous algorithm, the manipulator votes will always start with A, so the final score for A will be

$$s_A = 35000 + 30000 = 65000$$

The difference between the final score and the current scores of B, C, and D are 25000, 20000, and 5000 respectively.

Since we have imposed that the favorite candidate of the manipulator votes is A, these votes can only be

$$A > B > C > D$$
$$A > B > D > C$$
$$A > C > B > D$$
$$A > C > D > B$$
$$A > D > B > C$$
$$A > D > C > B.$$

We are looking for the number of votes of each type, that we denote α_1, α_2, α_3, α_4, α_5, α_6 respectively.

The system of equations then becomes

$$\alpha_1 \begin{pmatrix} 1 \\ 2/3 \\ 1/3 \\ 0 \end{pmatrix} + \alpha_2 \begin{pmatrix} 1 \\ 2/3 \\ 0 \\ 1/3 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1/3 \\ 2/3 \\ 0 \end{pmatrix} +$$
$$\alpha_4 \begin{pmatrix} 1 \\ 1/3 \\ 0 \\ 2/3 \end{pmatrix} + \alpha_5 \begin{pmatrix} 1 \\ 0 \\ 2/3 \\ 1/3 \end{pmatrix} + \alpha_6 \begin{pmatrix} 1 \\ 0 \\ 1/3 \\ 2/3 \end{pmatrix} \leq \begin{pmatrix} 30000 \\ 25000 \\ 20000 \\ 5000 \end{pmatrix}.$$

with equality for the first equation, under the constraints $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 \geq 0$ and $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 \in \mathbb{Z}$.

This system of equations can be analyzed for a standard mixed integer linear program using as objective function the function 0. We used the software R, in a i7 machine using Ubuntu 12.04 and the time it took to make the computation was 2 ms.

B. Any number of candidates

We can generalize the previous analysis to consider any number of candidates.

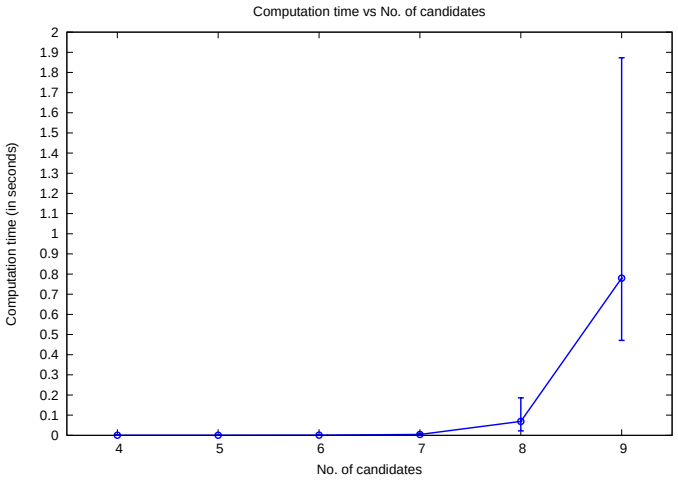
The algorithm is as follows:

Algorithm 1 ILP CCWM

1: Compute the combined votes of the S known votes, denoted u
2: Compute the score of the distinguished candidate p
3: Compute the differential between the score of the distinguished candidate and the combined votes of the S known votes, denoted $\hat{x}$
4: Compute all the permutations of votes in a matrix A
5: Create a linear program with $m!$ variables
6: Set the objective function to be zero
7: **repeat**
8: Add the constraint from the column of A and the vector $\hat{x}$
9: **until** we reach the last column of A
10: Set lower bound to be zero
11: Set the solution to be integer
12: Solve the integer linear program
13: Return the list of votes needed (if they exist)

Under the algorithm described above, we have done a Montecarlo simulation. The simulation consisted on a Borda election with increasing number of candidates (from 4 to 9 candidates) with 100 votes, 20 of which were manipulated and 80 truthful votes. We run the simulation 100 times and we present the mean, the minimum and the maximum computational time (in seconds) of the simulations (see Figure 1).

Fig. 1. Computational time (in seconds) vs No. of candidates.



V. Conclusions

In the present work, we have considered voting systems which can be expressed as scoring rules. In particular, we have analyzed the coalitional weighted manipulation problem and we have provided an integer linear programming approach to solve it. The approach allows us to solve in some milliseconds a problem which is known to be NP-hard.

Acknowledgment

The work presented in this paper has been partially carried out at LINCS (http://www.lincs.fr).

References

[1] A. Gibbard, "Manipulation of voting schemes: a general result", Econometrica 41 (4): 587-602, 1973.

[2] M. Satterthwaite, “Strategy-proofness and Arrow’s conditions: Existence and correspondance theorems for voting procedures and social welfare functions.” Journal of Economic Theory, 10:187–217, 1975.

[3] O. Lev and J. S. Rosenschein, “Convergence of Iterative Voting”, the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012), June 2012, Valencia, Spain, 611-618.

[4] M. Zuckerman, A. D. Procaccia, and J. S. Rosenschein, “Algorithms for the coalitional manipulation problem”, Journal of Artificial Intelligence, 173(2):392-412, February 2009.

[5] N. Betzler, R. Niedermeier, G. Woeginger, “Unweighted coalitional manipulation under the Borda rule is NP-hard”, Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), International Joint Conference on Artificial Intelligence.

[6] J. Bartholdi, C. A. Tovey, and M. A. Trick, “The computational difficulty of manipulating an election”, Social Choice and Welfare, 6:157–165, 1989.

[7] V. Conitzer, T. Sandholm, and J. Lang, “When are elections with few candidates hard to manipulate?”, Journal of the ACM, 54(3):1–33, 2007.