An Algorithm for the Coalitional Manipulation Problem under Maximin

Michael Zuckerman
michez@cs.huji.ac.il

Omer Lev
omerl@cs.huji.ac.il

Jeffrey S. Rosenschein
jeff@cs.huji.ac.il

The School of Computer Science and Engineering
The Hebrew University of Jerusalem

ABSTRACT

We introduce a new algorithm for the Unweighted Coalitional Manipulation problem under the Maximin voting rule. We prove that the algorithm gives an approximation ratio of $1\frac{2}{3}$ to the corresponding optimization problem. This is an improvement over the previously known algorithm that gave a 2-approximation. We also prove that its approximation ratio is no better than $1\frac{1}{2}$, i.e., there are instances on which a $1\frac{1}{2}$-approximation is the best the algorithm can achieve. Finally, we prove that no algorithm can approximate the problem better than to the factor of $1\frac{1}{2}$, unless $\mathcal{P} = \mathcal{NP}$.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms

Keywords

Social choice theory, Algorithms, Approximation

1. INTRODUCTION

In recent years, the importance of game-theoretic analysis as a formal foundation for multiagent systems has been widely recognized in the agent research community. As part of this research agenda, the field of *computational social choice* has arisen to explore ways in which multiple agents can effectively (and tractably) use elections to combine their individual, self-interested preferences into an overall choice for the group.

In an election, voters (agents) submit linear orders (rankings, or profiles) of the candidates (alternatives); a *voting rule* is then applied to the rankings in order to choose the winning candidate. In the prominent impossibility result proven by Gibbard and Satterthwaite [8, 11], it was shown that for any voting rule, a) which is not a dictatorship, b) which is onto the set of alternatives, and c) where there are at least three alternatives, there exist profiles where a voter can benefit by voting insincerely. Submitting insincere rankings in an attempt to benefit is called *manipulation*. Exploring the computational complexity of, and algorithms for, this *manipulation problem* is one of the most important research areas in computational social choice.

There are several ways to circumvent the Gibbard-Satterthwaite result, one of which is by using computational complexity as a barrier against manipulation. The idea behind this technique is as follows: although there may exist a successful manipulation, the voter must *discover* it before it can be used—but for certain voting rules, discovering a successful manipulation might be computationally hard. This argument was used already in 1989 by Bartholdi et al. [2], and in 1991 by Bartholdi and Orlin [1], where they proved, respectively, that second-order Copeland and Single Transferable Vote are both $\mathcal{NP}$-hard to manipulate.

Later, the complexity of coalitional manipulation was studied by Conitzer et al. [3]. In the coalitional manipulation problem, a coalition of potentially untruthful voters try to coordinate their ballots so as to make some preferred candidate win the election. Conitzer et al. studied the problem where manipulators are weighted: a voter with weight l counts as l voters, each of weight 1. This problem was shown to be $\mathcal{NP}$-hard, for many voting rules, even for a constant number of candidates. However, it has been argued that a more natural setting is the unweighted coalitional manipulation (UCM) problem, where all voters have equal power. In a recent paper [13], Xia et al. established as one of their main results that UCM is $\mathcal{NP}$-hard under the Maximin voting rule, even for 2 untruthful voters.

In 2009, Zuckerman et al. [14] defined a natural optimization problem for the unweighted setting (i.e., Unweighted Coalitional Optimization, UCO), namely finding the minimal number of manipulators sufficient to make some predefined candidate win. It is proven, as a corollary of their results, that the heuristic greedy algorithm proposed in the paper gives a 2-approximation to the UCO problem under Maximin. Here, we further study the UCO problem under Maximin, proposing a new greedy algorithm that gives a $1\frac{2}{3}$-approximation to the problem.[1] We then provide an example showing that the approximation ratio of the algorithm is no better than $1\frac{1}{2}$. Furthermore, since this gap (between $1\frac{2}{3}$ and $1\frac{1}{2}$) is due to the fact that the size of the manipulating coalition is rounded upwards, the actual bound on the ratio between the size of the coalition returned by the algorithm, and the minimum size of manipulating coalition, tends to $1\frac{1}{2}$ as the number of voters tends to infinity.

2. RELATED WORK

Behavior designed to alter outcomes in the Maximin voting rule has been widely studied. Perhaps the closest work to the UCM

[1]Strictly speaking, our algorithm is for the *decision* problem, but since the conversion of our algorithm to one for the optimization problem is straightforward, we consider it an approximation algorithm for the optimization problem.

Cite as: An Algorithm for the Coalitional Manipulation Problem under Maximin, Michael Zuckerman, Omer Lev and Jeffrey S. Rosenschein, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

problem is control by adding voters (AV), which has been studied by Faliszewski et al. [6]. The difference between AV control and UCM is that in the latter, manipulative voters can vote whatever they like in order to make their preferred candidate win, whereas in the former, the votes in the additional set are fixed. Faliszewski et al. proved that AV control in Maximin (as well as DV [Delete Voters] control and constructive AC [Add Candidates] control) is $\mathcal{NP}$-complete. In contrast, they showed polynomial-time algorithms for a combination of AC_u (a variant of Adding Candidates) and DC (Delete Candidates), and for a combination of destructive AC and DC control.

In another paper, Elkind et al. studied control of elections by cloning candidates [5]. For prominent voting rules (including Maximin) they characterized preference profiles for which there exist a successful cloning manipulation. For Maximin, a profile is manipulable by cloning if and only if the preferred candidate does not win, but is Pareto optimal. The authors also provided a simple linear-time algorithm for solving the cloning manipulation problem under Maximin.

Yet another topic that involves outcome-altering behavior in elections is bribery. In their paper [4], Elkind et al. investigated a model of bribery where the price of each vote depends on the amount of change that the voter is asked to implement. They showed that for their model, bribery is $\mathcal{NP}$-complete for Maximin, as well as for some other voting rules.

3. MAXIMIN VOTING, MANIPULATION

An election consists of a set $C = \{c_1, \ldots, c_m\}$ of candidates, and a set $S = \{v_1, \ldots, v_{|S|}\}$ of voters. Each voter provides a total order on the candidates (i.e., each voter submits a linear ranking of all the candidates). The setting also includes a *voting rule*, which is a function from the set of all possible combinations of votes to C.

The Maximin voting rule is defined as follows. For any two distinct candidates x and y, let $N(x, y)$ be the number of voters who prefer x over y. The *Maximin score* of x is $S(x) = \min_{y \neq x} N(x, y)$. The candidate with the highest Maximin score is the winner.

DEFINITION 3.1. *In the* CONSTRUCTIVE COALITIONAL UNWEIGHTED MANIPULATION (CCUM) *problem, we are given a set C of candidates, with a distinguished candidate $p \in C$, a set of (unweighted) voters S that have already cast their votes (these are the non-manipulators), and a set T of (unweighted) voters that have not yet cast their votes (these are the manipulators). We are asked whether there is a way to cast the votes in T so that p wins the election.*

DEFINITION 3.2. *In the* UNWEIGHTED COALITIONAL OPTIMIZATION (UCO) *problem we are given a set C of candidates, with a distinguished candidate $p \in C$, and a set of (unweighted) voters S that have already cast their votes (the non-manipulators). We are asked for the* minimal *n such that a set T of size n of (unweighted) manipulators can cast their votes in order to make p win the election.*

REMARK 3.3. *We implicitly assume here that the manipulators have full knowledge about the non-manipulators' votes (this is the common assumption in the literature). Unless explicitly stated otherwise, we also assume that ties are broken adversarially to the manipulators, so that if p ties with another candidate, p loses. The latter assumption is equivalent to formulating the manipulation problems in their* unique winner *version, when one assumes that all candidates with maximal score win, but asks that p be the only winner.*

Throughout this paper we will use the convention, unless explicitly stated otherwise, that $|C| = m$, $|S| = N$ and $|T| = n$. We will denote $N_i(x, y) = |\{j \mid x \succ_j y, \succ_j \in S \cup \{1, \ldots, i\}\}|$. That is, $N_i(x, y)$ will denote the number of voters from S and from the first i voters of T that prefer x over y (assuming S is fixed, and fixing some order on the voters of T). Furthermore, we will denote by $S_i(c)$ the accumulated score of candidate c from the voters of S and the first i voters of T. By definition, for each $x \in C$, $S_i(x) = \min_{y \neq x} N_i(x, y)$. Also, we denote for $x \in C$, $\text{MIN}_i(x) = \{y \in C \setminus \{x\} \mid S_i(x) = N_i(x, y)\}$. We denote for $0 \leq i \leq n$, $ms(i) = \max_{c \in C \setminus \{p\}} S_i(c)$. That is, $ms(i)$ is the maximum score of the opponents of p after i manipulators have voted.

DEFINITION 3.4. *The* Condorcet winner *of an election is the candidate who, when compared with every other candidate, is preferred by more voters.*

Next we give a lower bound on the approximation ratio of any polynomial-time algorithm for the UCO problem under Maximin.

PROPOSITION 3.5. *No polynomial-time algorithm approximating the UCO problem under Maximin can do better than $1\frac{1}{2}$, unless $\mathcal{P} = \mathcal{NP}$.*

PROOF. Suppose, for contradiction, that there exists a polynomial-time approximation algorithm $\mathcal{A}$ to the UCO problem under Maximin having approximation ratio $r < 1\frac{1}{2}$. Then when $opt = 2$, the minimal size of manipulating coalition returned by $\mathcal{A}$ is $n \leq r \cdot opt < 3$. Since the size of the coalition is an integer, it follows that $n = 2$. Therefore, $\mathcal{A}$ can decide the CCUM problem for the coalition of 2 manipulators, which contradicts the fact that this problem is $\mathcal{NP}$-complete [13] (unless $\mathcal{P} = \mathcal{NP}$). □

4. THE ALGORITHM

Our algorithm for the CCUM problem under the Maximin voting rule is given as Algorithm 1 (see the final page of the paper). The intuition behind Algorithm 1 is as follows. The algorithm tries in a greedy manner to maximize the score of p, and to minimize the scores of p's opponents. To achieve this, for all i, manipulator i puts p first in his preference list, making the score of p grow by 1. He then builds a digraph $G^{i-1} = (V, E^{i-1})$, where $V = C \setminus \{p\}$, $(x, y) \in E^{i-1}$ iff ($y \in \text{MIN}_{i-1}(x)$ and $p \notin \text{MIN}_{i-1}(x)$). He tries first to rank candidates without any outgoing edges from them, since their score will not grow this way (because their score is achieved vs. candidates who were already ranked above them). When there are no candidates without outgoing edges, the algorithm tries to find a cycle with two adjacent vertices having the lowest score. If it finds such a cycle, then it picks the front vertex of these two. Otherwise, any candidate with the lowest score is chosen. After ranking each candidate, the edges in the graph are updated, so that all candidates whose minimal candidate has already been ranked will be with outgoing degree 0. For an edge (x, y), if y has already been ranked, we remove all the edges going out of x, since if we rank x now, its score will not go up, and so it does not depend on other candidates in $\text{MIN}_{i-1}(x)$. There is no need of an edge (x, y) if $p \in \text{MIN}_{i-1}(x)$, since for all $x \in C \setminus \{p\}$, p is always ranked above x, and so whether y is ranked above x or not, the score of x will not grow.

Let us note a few points regarding the algorithm:

- When picking a candidate with an out-degree 0, the algorithm first chooses candidates with the lowest score (among the candidates with an out-degree 0). It appears that this issue is critical for getting the approximation ratio of $1\frac{2}{3}$.

- The candidates with out-degree 0 are kept in stacks in order to guarantee a DFS-like order among candidates with the same score (this is needed for Lemma 6.4, below, to work).
- After a candidate b is added to the manipulator's preference list, for each candidate y who has an outgoing edge (y, b), the algorithm removes all the outgoing edges of y, puts it into the appropriate stack, and assigns b to be y's "father". Essentially, the assignment $y.father \leftarrow b$ means that due to b the score of y did not grow. The "father" relation is used to analyze the algorithm.
- Note the subtle difference between calculating the scores in Algorithm 1 in this paper, as compared to Algorithm 1 in [14]. In the latter, the manipulator i calculates what the score would be of the current candidate x if he put x at the current place in his preference list; in the algorithm we are now presenting, manipulator i just calculates $S_{i-1}(x)$. This difference is due to the fact that here, when we calculate the score of x, we know whether $d_{out}(x) > 0$, i.e., we know whether the score of x will grow by 1 if we put it at the current available place. So we separately compare the scores of candidates with out-degree > 0, and the scores of candidates with out-degree 0.

DEFINITION 4.1. *We refer to an iteration of the main* **for** *loop in lines 3–37 of Algorithm 1 as a* stage *of the algorithm. That is, a stage of the algorithm is a vote of any manipulator.*

DEFINITION 4.2. *In the digraph G^i built by the algorithm, if there exists an edge (x, y), we refer to $N_i(x, y) = S_i(x)$ as the* weight *of the edge (x, y).*

5. 2-APPROXIMATION

We first prove that Algorithm 1 has an approximation ratio of 2. We then use this result in the subsequent proof of the $1\frac{2}{3}$ approximation ratio.

THEOREM 5.1. *Algorithm 1 has a 2-approximation ratio for the UCO problem under the Maximin voting rule.*

To prove the above theorem, we first need the following two lemmas. In the first lemma, we prove that a certain sub-graph of the graph built by the algorithm contains a cycle passing through some distinguished vertex. We first introduce some more notation.

Let $G^i = (V, E^i)$ be the directed graph built by Algorithm 1 in stage $i + 1$. For a candidate $x \in C \setminus \{p\}$, let $G^i_x = (V^i_x, E^i_x)$ be the graph G^i reduced to the vertices that were ranked below x in stage $i + 1$, including x.

Let $V^i(x) = \{y \in V^i_x \mid \text{there is a path in } G^i_x \text{ from } x \text{ to } y\}$. Also, let $G^i(x)$ be the sub-graph of G^i_x induced by $V^i(x)$.

LEMMA 5.2. *Let i be an integer, $0 \le i \le n - 1$. Let $x \in C \setminus \{p\}$ be a candidate. Denote $t = ms(i)$. Suppose that $S_{i+1}(x) = t + 1$. Then $G^i(x)$ contains a cycle passing through x.*

PROOF. First of all note that for all $c \in V^i(x)$, $S_i(c) = t$. It follows from the fact that by definition $S_i(c) \le t$. On the other hand, $S_i(x) = t$, and all the other vertices in $V^i(x)$ were ranked below x. Together with the fact that the out-degree of x was greater than 0 when x was picked, it gives us that for all $c \in V^i(x)$, $S_i(c) \ge t$, and so for all $c \in V^i(x)$, $S_i(c) = t$. We claim that for all $c \in V^i(x)$, $\text{MIN}_i(c) \subseteq V^i(x)$. If, by way of contradiction, there exists $c \in V^i(x)$ s.t. there is $b \in \text{MIN}_i(c)$ where $b \notin V^i(x)$, then $b \notin V^i_x$, since otherwise, if $b \in V^i_x$, then from $c \in V^i(x)$ and $(c, b) \in E^i_x$ we get that $b \in V^i(x)$. So $b \notin V^i_x$, which means that b was ranked by $i + 1$ above x. After we ranked b we removed all the outgoing edges from c, and so we chose c before x since $d_{out}(c) = 0$ and $d_{out}(x) > 0$ (since the score of x increased in stage $i + 1$). This contradicts the fact that $c \in V^i(x) \subseteq V^i_x$. Therefore, for every vertex $c \in V^i(x)$ there is at least one edge in $G^i(x)$ going out from c. Hence, there is at least one cycle in $G^i(x)$. Since at the time of picking x by voter $i + 1$, for all $c \in V^i(x)$, $d_{out}(c) > 0$, and by the observation that for all $c \in V^i(x)$, $S_i(c) = t$, we have that the algorithm picked the vertex x from a cycle (lines 21–22 of the pseudocode). □

In the following lemma, we show an upper bound on the growth rate of the scores of p's opponents.

LEMMA 5.3. *For all $0 \le i \le n - 2$, $ms(i + 2) \le ms(i) + 1$.*

PROOF. Let $0 \le i \le n - 2$. Let $x \in C \setminus \{p\}$ be a candidate. Denote $t = ms(i)$. By definition, $S_i(x) \le t$. We would like to show that $S_{i+2}(x) \le t + 1$. If $S_{i+1}(x) \le t$, then $S_{i+2}(x) \le S_{i+1}(x) + 1 \le t + 1$, and we are done. So let us assume now that $S_{i+1}(x) = t + 1$.

Let $V^i(x)$ and $G^i(x)$ be as before. By Lemma 5.2, $G^i(x)$ contains at least one cycle. Let U be one such cycle. Let $a \in U$ be the vertex that was ranked highest among the vertices of U in stage $i + 1$. Let b be the vertex before a in the cycle: $(b, a) \in U$. Since b was ranked below a in stage $i + 1$, it follows that $S_{i+1}(b) = S_i(b) \le t$.

Suppose, for contradiction, that $S_{i+2}(x) > t+1$. Then the score of x increased in stage $i + 2$, and so when x was picked by $i + 2$, its out-degree in the graph was not 0. x was ranked by $i + 2$ at place s^*. Then b was ranked by $i + 2$ above s^*, since otherwise, when we had reached the place s^*, we would not pick x since b would be available (with out-degree 0, or otherwise—with score $S_{i+1}(b) \le t < t + 1 = S_{i+1}(x)$)—a contradiction.

Denote by Z_1 all the vertices in $V^i(x)$ that have an outgoing edge to b in $G^i(x)$. For all $z \in Z_1$, $b \in \text{MIN}_i(z)$, i.e., $S_i(z) = N_i(z, b)$. We claim that all $z \in Z_1$ were ranked by $i + 2$ above x. If, by way of contradiction, there is $z \in Z_1$, s.t. until the place s^* it still was not added to the preference list, then two cases are possible:

1. If $(z, b) \in E^{i+1}$, then after b was added to $i+2$'s preference list, we removed all the outgoing edges of z, and we would put in z (with out-degree 0) instead of x, a contradiction.
2. $(z, b) \notin E^{i+1}$. Since $(z, b) \in E^i$, we have $S_i(z) = N_i(z, b)$. Also since z was ranked by $i + 1$ below x, it follows that $S_i(z) = t$. So from $(z, b) \notin E^{i+1}$, we have that $S_{i+1}(z) = t$ and $N_{i+1}(z, b) = t + 1$. Therefore, when reaching the place s^* in the $i + 2$'s preference list, whether $d_{out}(z) = 0$ or not, we would not pick x (with the score $S_{i+1}(x) = t+1$) since z (with the score $S_{i+1}(z) = t$) would be available, a contradiction.

Denote by Z_2 all the vertices in $V^i(x)$ that have an outgoing edge in $G^i(x)$ to some vertex $z \in Z_1$. In the same manner we can show that all the vertices in Z_2 were ranked in stage $i + 2$ above x. We continue in this manner, by defining sets $Z_3, \ldots$, where the set Z_l contains all vertices in $V^i(x)$ that have an outgoing edge to some vertex in Z_{l-1}; the argument above shows that all elements of these sets are ranked above x in stage $i + 2$. As there is a path from x to b in $G^i(x)$, we will eventually reach x in this way, i.e., there is some l such that Z_l contains a vertex y, s.t. $(x, y) \in E^i(x)$.

Now, if $(x, y) \in E^{i+1}(x)$, then since y was ranked by $i + 2$ above x, we have $S_{i+2}(x) = S_{i+1}(x) = t + 1$, a contradiction.

And if $(x,y) \notin E^{i+1}(x)$, then since $(x,y) \in E^i(x)$ we get that $N_{i+1}(x,y) = t+1$ and $S_{i+1}(x) = t$, a contradiction. □

We are now ready to prove Theorem 5.1.

PROOF OF THEOREM 5.1. Let *opt* denote the minimum size of coalition needed to make p win. It is easy to see that $opt \geq ms(0) - S_0(p) + 1$. We set $n = 2ms(0) - 2S_0(p) + 2 \leq 2opt$. Then, by Lemma 5.3:

$$ms(n) \leq ms(0) + \left\lceil \frac{n}{2} \right\rceil = 2ms(0) - S_0(p) + 1.$$

Whereas:

$$S_n(p) = S_0(p) + n = 2ms(0) - S_0(p) + 2 > ms(n).$$

So p will win when the coalition of manipulators is of size n. □

6. $1\frac{2}{3}$-APPROXIMATION

Our next goal is to prove that Algorithm 1 has an approximation ratio of $1\frac{2}{3}$ when there are no 2-cycles in the graphs built by the algorithm.

THEOREM 6.1. *For instances where there are no 2-cycles in the graphs G^i built by Algorithm 1, it gives a $1\frac{2}{3}$-approximation of the optimum.*

Let us give a general short overview of the proof of the above theorem (we will give an intuitive description rather than a formal/rigorous one). In Lemmas 6.2–6.5 we aim to prove that the maximum score of p's opponents grows 3 times slower than the score of p, at the most. After proving this, the theorem will easily follow. Recall that we proved in Lemma 5.2 that there is a cycle passing through x after i stages. Then we prove that at least one such cycle stays after stage $i+1$ (Lemma 6.2). In this cycle there are 2 consecutive vertices with a low score ($= t$) (Lemma 6.3). During stage $i+2$ only the score of one of them will increase (at the most), so the score of the second one will remain t (Lemma 6.4). Then, in stage $i+3$ this second vertex will be ranked above x, and the score of x will not grow (and remain $t+1$) (Lemma 6.5). This way, during 3 stages the score of x increases only by 1, whereas the score of p grows by 1 every stage.

Let us now state and prove the lemmas more formally.

LEMMA 6.2. *Let $x \in C \setminus \{p\}$ be a candidate such that $S_{i+1}(x) = t+1$ (where $t = ms(i)$). Let $G^i(x)$ be as before. Then at least one cycle in $G^i(x)$ that passes through x will stay after stage $i+1$, i.e., in G^{i+1}.*

PROOF. In Lemma 5.2 we have proved that in $G^i(x)$ at least one cycle passes through x. Since x appears in the preference list of $i+1$ above all the $\text{MIN}_i(x)$, it follows that each edge going out of x in $G^i(x)$, stays also in G^{i+1}. After we added x to the preference list of $i+1$, all the vertices in all the cycles passing through x were added in some order to the preference list of $i+1$, while they were with out-degree 0 at the time they were picked (it can be proved by induction on the length of the path from the vertex to x). Therefore, their "father" field was not null when they were picked. We have to prove that there is at least one cycle whose vertices were added in the reverse order (and then all the edges of the cycle stayed in G^{i+1}). Let $z_1 \in C \setminus \{p, x\}$ be some vertex such that $(x, z_1) \in G^i(x)$ and there is a path in $G^i(x)$ from z_1 to x. Let $z_2 = z_1.father$. As observed earlier, $z_2 \neq null$. We first show that when z_2 was picked by $i+1$, it was with out-degree 0. Indeed, if, by contradiction, we suppose otherwise, then z_2 would have been picked after z_1 (the proof is by induction on the length of the shortest path from vertex to x, that each vertex such that there is a path from it to x was picked before z_2), and this is a contradiction to the fact that $z_2 = z_1.father$. Therefore, the "father" field of z_2 after stage $i+1$ is not null.

Let $z_3 = z_2.father$. If $z_3 = x$ then we are done because we have found a cycle $x \to z_1 \to z_2 \to z_3 = x$ which was ranked in stage $i+1$ in the reverse order. Otherwise, by the same argument as before, we can show that when z_3 was picked, its out-degree was 0. This way we can pass from a vertex to its father until we reach p or null. We now show that we cannot reach p this way. Indeed, if, by contradiction, we reach p, then there is a path from x to p in G^i, and so all the vertices in this path, including x, were picked when their out-degree was 0, and this is a contradiction to the fact that the score of x went up in stage $i+1$. Therefore, we cannot reach p when we go from a vertex to its father starting with z_1. Now, let z_j be the last vertex before null in this path. We would like to show that $z_j = x$. If, by contradiction, z_j was picked before x by voter $i+1$, then all the vertices $z_{j-1}, \ldots, z_2, z_1$ would have been picked before x, when their out-degree is 0, and then x would have been picked when its out-degree is 0. This is a contradiction to the fact that x's score increased in stage $i+1$. Now suppose by contradiction that z_j was picked after x in stage $i+1$. Then all the vertices that have a path from them to x, including z_1, would have been picked before z_j in stage $i+1$, since the out-degree of z_j was greater than 0 when it was picked. This is a contradiction to the fact that z_j was picked before z_1. So, $z_j = x$. This way we got a cycle $x \to z_1 \to \ldots \to z_{j-1} \to x$ which was ranked in the reverse order in stage $i+1$. □

LEMMA 6.3. *Suppose that there are no 2-cycles in the graphs built by the algorithm. Let $x \in C \setminus \{p\}$ be a candidate such that $S_{i+1}(x) = t+1$ (where $t = ms(i)$), and let $G^i(x)$ be as described before Lemma 5.2. For each cycle U in $G^i(x)$, if U exists in G^{i+1}, i.e., after stage $i+1$, then there are 3 distinct vertices a, b, c, s.t. $(c,b) \in U$, $(b,a) \in U$ and $S_{i+1}(b) = N_{i+1}(b,a) = S_{i+1}(c) = N_{i+1}(c,b) = t$.*

PROOF. Let $U \subseteq E^i(x)$ be a cycle which stays also after $i+1$ stages. Let a be the vertex which in stage $i+1$ was chosen first among the vertices of U. Let b be the vertex before a in U, i.e., $(b,a) \in U$, and let c be the vertex before b in U, i.e., $(c,b) \in U$. Since there are no 2-cycles, a, b, c are all distinct vertices. Recall that for each $y \in V^i(x)$, $S_i(y) = t$. Since b was ranked below a in stage $i+1$, we have $S_{i+1}(b) = N_{i+1}(b,a) = N_i(b,a) = S_i(b) = t$. If c was chosen after b in stage $i+1$, then $S_{i+1}(c) = N_{i+1}(c,b) = N_i(c,b) = t$ and we are done. We now show that c cannot be chosen before b in stage $i+1$. If, by way of contradiction, c were chosen before b, since after ranking a, $d_{out}(b) = 0$, it follows that when c was picked, its out-degree was also 0. Hence, there exists $d \in \text{MIN}_i(c)$ which was picked by $i+1$ before c. And so, $S_{i+1}(c) = t$. On the other hand, since c was picked before b, we have $N_{i+1}(c,b) = t+1 > S_{i+1}(c)$, and so the edge (c,b) does not exist in G^{i+1}, a contradiction to the fact that the cycle U stayed after stage $i+1$. □

LEMMA 6.4. *Suppose that there are no 2-cycles in the graphs built by the algorithm. Let $x \in C \setminus \{p\}$ be a candidate such that $S_{i+1}(x) = t+1$ (where $t = ms(i)$). Then after stage $i+2$ at least one of the following will hold:*

1. *There will be a vertex w in G^{i+2} s.t. $p \in MIN_{i+2}(w)$ and there will be a path from x to w.*
2. *There will be a vertex w in G^{i+2} with $S_{i+2}(w) \leq t$, s.t. there will be a path from x to w.*

PROOF. 1. If there is a vertex w s.t. $p \in \text{MIN}_{i+1}(w)$ and there is a path from x to w in G^{i+1}, w.l.o.g. let us assume that w was picked first in stage $i+2$ among all such vertices. It is easy to see that $p \in \text{MIN}_{i+2}(w)$. If $x = w$, then trivially condition 1 holds, and we are done. Otherwise, in stage $i+2$, w was ranked above x. Let us build a chain of vertices, starting from x, by passing from a vertex to its father, as was assigned in the stage $i+2$. The chain stops when we reach p or null. If we reach p this way then we are done, because $a = b.father$ means that there is an edge (b, a) in G^{i-1}, and it stayed in G^{i+2} (because a was ranked above b). Now we show that we can't reach null this way. Suppose, for contradiction, that we reach null, and let z be the vertex before null in the chain. If z was ranked above w in stage $i+2$, then we get a contradiction since at the time of ranking z, $d_{out}(w) = 0$, whereas $d_{out}(z) > 0$, and we would prefer w over z. On the other hand, if w was ranked above z, then x should have been ranked above z too, since there is a path in G^{i+1} from x to w whereas $d_{out}(z) > 0$. So we got a contradiction since, by definition, z was ranked above x.

2. Now suppose that the condition in the first item does not hold. If there is a vertex w, s.t. $S_{i+1}(w) < t$ and there is a path in G^{i+1} from x to w, again, w.l.o.g. let us assume that w was picked first in stage $i+2$ among all such vertices. Then $S_{i+2}(w) \leq t$, and similarly to item 1 above, there is a path in G^{i+2} from x to w.

   Now let us suppose that the above conditions do not hold. Let us look at the vertex y which in stage $i+2$ was picked first from a cycle U s.t. there is a path from x to U, and there are two consecutive edges in U, each with weight t. By Lemma 6.3 and Lemma 6.2 such a vertex y exists. According to the algorithm (lines 21–22) and to the definition of y, before y only vertices s.t. there is no path from x to them, could be picked. Therefore, there is no path from y to earlier-picked vertices. So when y was picked, its out-degree was > 0, and hence all the edges going out of y stayed after stage $i+2$. According to the algorithm (lines 21–22), there is a vertex w s.t. $S_{i+1}(w) = N_{i+1}(w, y) = t$ and there is a path from y to w in G^{i+1}. Let W be the set of all such vertices w. According to the algorithm (lines 17–18), in stage $i+2$, all the vertices in W will be picked before all the vertices z with $S_{i+1}(z) = N_{i+1}(z, y) = t+1$.

   Now let us go from a vertex to its father, starting with x (like in Lemma 6.2) till we reach null (we cannot reach p this way since condition 1 does not hold). Similarly to Lemma 6.2, it can be verified that the last vertex before null is y. If we passed this way through some vertex $w \in W$ then we are done, since we got a path from x to w, and $S_{i+2}(w) = t$ (because w was picked in stage $i+2$ after y). Otherwise we are in the next situation: the path from x to U, connects to U through the vertex y (if this is not the case then we will pass through some $w \in W$ since according to the algorithm (lines 16–18 and 32), all the vertices which have a path from them to w will be picked before all other vertices w' with an edge (w', y) with weight $t+1$ and a path from y to w'). Let b be a vertex s.t. $(y, b) \in G^{i+1}$ (and so also $(y, b) \in G^{i+2}$) and there is a path in G^{i+1} from b to some $w \in W$. Since b belongs to a cycle with two edges of the weight t and there is a path from x to b, it follows that b was picked by $i+2$ after y. As there is a path from b to w, it follows that b was picked when $d_{out}(b) = 0$, and hence $b.father \neq null$. Like in Lemma 6.2, we go from a vertex to its father, starting with b, until we reach w. This way we got a path in G^{i+2} from x through y and b to w, and as mentioned earlier, $S_{i+2}(w) = t$.

□

The next lemma is central in the proof of Theorem 6.1. It states that the maximum score of p's opponents grows rather slowly.

LEMMA 6.5. *If there are no 2-cycles in the graphs built by the algorithm, then for all i, $0 \leq i \leq n-3$ it holds that $ms(i+3) \leq ms(i)+1$.*

PROOF. Let i, $0 \leq i \leq n-3$. Let $x \in C \setminus \{p\}$ be a candidate. Denote $ms(i) = t$. We need to prove that $S_{i+3}(x) \leq t+1$. If $S_{i+1}(x) \leq t$, then similarly to Lemma 5.3 we can prove that $S_{i+3}(x) \leq t+1$. So now we assume that $S_{i+1}(x) = t+1$. By Lemma 5.3, we have that $S_{i+2}(x) = t+1$. Suppose by contradiction that $S_{i+3}(x) = t+2$. x was ranked in stage $i+3$ at the place s^*. By Lemma 6.4 there exists a vertex w s.t. there is a path in G^{i+2} from x to w, and $p \in \text{MIN}_{i+2}(w)$ or $S_{i+2}(w) \leq t$. Then w was ranked in stage $i+3$ above the place s^*, because the score of x increased in stage $i+3$, and if, by contradiction, w was not ranked above the place s^*, then when we got to the place s^* we would prefer w over x. It is easy to see that all the vertices that have a path in G^{i+2} from them to w, and which were ranked below w in stage $i+3$, did not have their scores increased in that stage (since we took them one after another in the reverse order on their path to w when they were with out-degree 0). And as x was ranked below w, its score did not increase as well, and so $S_{i+3}(x) = S_{i+2}(x) = t+1$, a contradiction. □

LEMMA 6.6. *If the minimum number of manipulators needed to make p win is equal to 1, then Algorithm 1 performs optimally, i.e., finds the manipulation for $n = 1$.*

PROOF. Let us denote by *opt* the minimum number of manipulators needed to make p win the election. Let $S_i^*(a)$ denote the score of $a \in C$ after i manipulators voted in the optimal algorithm, and let $ms^*(i)$ be the maximum score of p's opponents after i manipulators voted in the optimal algorithm. Assume that $opt = 1$. If $S_0(p) > ms(0)$ then $opt = 0$, a contradiction. On the other hand, if $S_0(p) < ms(0)$, then $S_1^*(p) \leq ms(0) \leq ms^*(1)$, so p is not a unique winner after the manipulator voted, a contradiction. Therefore, $S_0(p) = ms(0)$. Also, $S_1^*(p) = ms(0)+1$ and $ms^*(1) = ms(0)$ (otherwise, p would not be a unique winner of the election). We need to show that $ms(1) = ms(0)$. Let $x \in C \setminus \{p\}$. If $S_0(x) < ms(0)$ then trivially $S_1(x) \leq ms(0)$ and we are done. Now suppose that $S_0(x) = ms(0)$. Suppose, by contradiction, that $S_1(x) = ms(0)+1$. Then when x was ranked by the first manipulator, $d_{out}(x) > 0$. Denote, as before, by V_x^0 the candidates that were ranked by Algorithm 1 below x, including x, in stage 1. For each $y \in V_x^0$, $S_0(y) = ms(0)$ and $d_{out}(y) > 0$. Therefore, if we put y instead of x, its score will increase to $ms(0)+1$. Let $b \in V_x^0$ be the candidate ranked highest among candidates in V_x^0 by the optimal algorithm. Then $ms^*(1) \geq S_1^*(b) = ms(0)+1$, contradicting the fact that $ms^*(1) = ms(0)$. □

We are now ready to prove the main theorem.

PROOF OF THEOREM 6.1. Let *opt* be as before. It is easy to see that $opt \geq ms(0) - S_0(p) + 1$. We shall prove first that Algorithm 1 will find a manipulation for $n = \left\lceil \frac{3ms(0) - 3S_0(p) + 3}{2} \right\rceil \leq \lceil \frac{3}{2} opt \rceil$. And indeed, by Lemma 6.5,

$$ms(n) \leq ms(0) + \left\lceil \frac{n}{3} \right\rceil = ms(0) + \left\lceil \frac{ms(0) - S_0(p) + 1}{2} \right\rceil.$$

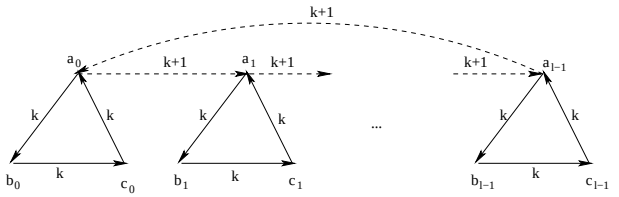

Figure 1: Example for lower bound on approximation ratio

Whereas,

$$\begin{aligned}
S_n(p) &= S_0(p) + n \\
&= S_0(p) + (ms(0) - S_0(p) + 1) + \left\lceil \frac{ms(0) - S_0(p) + 1}{2} \right\rceil \\
&= ms(0) + 1 + \left\lceil \frac{ms(0) - S_0(p) + 1}{2} \right\rceil \\
&> ms(0) + \left\lceil \frac{ms(0) - S_0(p) + 1}{2} \right\rceil \\
&\geq ms(n).
\end{aligned}$$

Now, by Lemma 6.6, when $opt = 1$, the algorithm performs optimally (i.e., finds the manipulation for $n = 1$). We have just proved that for $opt = 2$ the algorithm finds the manipulation when $n \leq \lceil \frac{3}{2} opt \rceil = 3$. When $opt = 3$, the algorithm finds the manipulation when $n \leq \lceil \frac{3}{2} opt \rceil = 5$. It is easy to see that for all $opt > 3$, $\lceil \frac{3}{2} opt \rceil < \frac{5}{3} opt$. Therefore, the approximation ratio of Algorithm 1 is $\leq \frac{5}{3} = 1\frac{2}{3}$. □

It is worth noting that from the proof above we have that as *opt* tends to infinity, the bound on the ratio between the size of the manipulating coalition returned by Algorithm 1 and *opt* tends to $1\frac{1}{2}$, since n is bounded by $\lceil 1\frac{1}{2} opt \rceil$.

THEOREM 6.7. *The $1\frac{2}{3}$-approximation ratio of Algorithm 1 is valid also when there are 2-cycles in the graphs built by the algorithm.*

We omit the proof of the above theorem due to space limitations.

7. LOWER BOUND ON THE APPROXIMATION RATIO OF THE ALGORITHM

THEOREM 7.1. *There is an asymptotic lower bound of $1\frac{1}{2}$ to the approximation ratio of Algorithm 1.*

PROOF. Consider the following example (see Figure 1). Let $m = |C|$ be of the form $m = 3^t + 1$ for an integer $t \geq 2$. Denote $l = 3^{t-1} = \frac{m-1}{3}$. Let $C = \{p, a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_{l-1}, b_{l-1}, c_{l-1}\}$. Let N be a multiple of 3, $N \geq 6$. Let $k = \frac{N}{3}$. $S_0(p) = 0$; for all $j, 0 \leq j \leq l-1$: $S_0(a_j) = N_0(a_j, b_j) = S_0(b_j) = N_0(b_j, c_j) = S_0(c_j) = N_0(c_j, a_j) = k$. In addition, for each $j, 0 \leq j \leq l-2$: $N_0(a_j, a_{j+1}) = k+1$, and $N_0(a_{l-1}, a_0) = k+1$. We first show that there exists a profile of non-manipulators that induces the above scores. We will have non-manipulator voters of 6 types; $\frac{N-3}{3}$ voters of each of the types (1), (2) and (3), and one voter of each of the types (4), (5) and (6). In all types, p is ranked in last place. To conserve space, we denote by A_j the fragment $a_j \succ c_j \succ b_j$ of the preference, by B_j the fragment $b_j \succ a_j \succ c_j$, and by C_j the fragment $c_j \succ b_j \succ a_j$. When showing the preference lists of the voters, it is convenient to use the trinary representation of the indices. We have candidates $\{p, a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_{22\ldots2}, b_{22\ldots2}, c_{22\ldots2}\}$. For preference list of type (1), we define the order $0 \succ_1 2 \succ_1 1$. In this preference list, we have the fragments A_j ordered by the order $\succ_1$, and p is at the end. In type (2), we have the fragments C_j ordered by the order $2 \succ_2 1 \succ_2 0$, with p at the end. In type (3), we have the fragments B_j ordered by the order $1 \succ_3 0 \succ_3 2$, with p at the end. In type (4), we have the fragments A_j ordered by $0 \succ_4 1 \succ_4 2$, with p at the end. In type (5), we have the fragments C_j ordered by $1 \succ_5 2 \succ_5 0$, with p at the end. Finally, in type (6) there are the fragments B_j ordered by $2 \succ_6 0 \succ_6 1$, with p at the end.

For instance, the next example illustrates the above profile for $t = 3$ ($m = 28$), and it easily generalizes to any $t \geq 2$.

$$\begin{aligned}
&(1){:}A_0 \succ A_2 \succ A_1 \succ A_{20} \succ A_{22} \succ A_{21} \succ A_{10} \succ A_{12} \succ A_{11} \succ p \\
&(2){:}C_{22} \succ C_{21} \succ C_{20} \succ C_{12} \succ C_{11} \succ C_{10} \succ C_2 \succ C_1 \succ C_0 \succ p \\
&(3){:}B_{11} \succ B_{10} \succ B_{12} \succ B_1 \succ B_0 \succ B_2 \succ B_{21} \succ B_{20} \succ B_{22} \succ p \\
&(4){:}A_0 \succ A_1 \succ A_2 \succ A_{10} \succ A_{11} \succ A_{12} \succ A_{20} \succ A_{21} \succ A_{22} \succ p \\
&(5){:}C_{11} \succ C_{12} \succ C_{10} \succ C_{21} \succ C_{22} \succ C_{20} \succ C_1 \succ C_2 \succ C_0 \succ p \\
&(6){:}B_{22} \succ B_{20} \succ B_{21} \succ B_2 \succ B_0 \succ B_1 \succ B_{12} \succ B_{10} \succ B_{11} \succ p
\end{aligned}$$

It could be verified that the graph G^0 which matches the above profile looks as in Figure 1 (we omitted some of the dotted edges).

Now we will show that for the above example the approximation ratio of the algorithm is at least $1\frac{1}{2}$. Consider the following preference list of the manipulators:

$$\begin{aligned}
&p \succ A_{l-1} \succ A_{l-2} \succ \ldots \succ A_0 \\
&p \succ A_{l-2} \succ A_{l-3} \succ \ldots \succ A_0 \succ A_{l-1} \\
&p \succ A_{l-3} \succ A_{l-4} \succ \ldots \succ A_0 \succ A_{l-1} \succ A_{l-2} \\
&\ldots
\end{aligned}$$

It can be verified that in the above preference list, the maximum score of p's opponents ($ms(i)$) grows by 1 every $\frac{m-1}{3}$ stages (starting with $k+1$). In addition, p's score grows by 1 every stage. Therefore, when we apply the voting above, the minimum number of stages (manipulators) n^* needed to make p win the election should satisfy $n^* > k + 1 + \lceil \frac{3n^*}{m-1} \rceil$. Since $\lceil \frac{3n^*}{m-1} \rceil < \frac{3n^*}{m-1} + 1$, the sufficient condition for making p win is:

$$n^* > k + 1 + \frac{3n^*}{m-1} + 1.$$

So, we have,

$$\begin{aligned}
(m-1)n^* &> (m-1)(k+2) + 3n^* \\
(m-4)n^* &> (m-1)(k+2) \\
n^* &> \frac{(m-1)(k+2)}{m-4}.
\end{aligned}$$

For large-enough m, $\frac{(m-1)(k+2)}{m-4} < k+3$, and so $n^* = k+3$ would be enough to make p win the election.

Now let us examine what Algorithm 1 will do when it gets this example as input. One of the possible outputs of the algorithm looks like this:

$$\begin{aligned}
&p \succ C_0 \succ C_1 \succ \ldots \succ C_{l-1} \\
&p \succ B_1 \succ B_2 \succ \ldots \succ B_{l-1} \succ B_0 \\
&p \succ A_2 \succ A_3 \succ \ldots \succ A_{l-1} \succ A_0 \succ A_1 \\
&p \succ C_3 \succ C_4 \succ \ldots \succ C_{l-1} \succ C_0 \succ C_1 \succ C_2 \\
&\ldots
\end{aligned}$$

It can be verified that in the above preference list, $ms(i)$ grows by 1 every 3 stages, and p's score grows by 1 every stage. Therefore, the number of stages n returned by Algorithm 1 that are needed to make p win the election satisfies $n > k + \lceil \frac{n}{3} \rceil$. Since $\lceil \frac{n}{3} \rceil \geq \frac{n}{3}$, the necessary condition for making p win the election is:

$$n > k + \frac{n}{3}.$$

And then we have,

$$\begin{aligned} 3n &> 3k + n \\ 2n &> 3k \\ n &> \frac{3}{2}k \end{aligned}$$

So we find that the ratio $\frac{n}{n^*}$ tends to $1\frac{1}{2}$ as m and N (and k) tend to infinity. □

8. DISCUSSION

In spite of the popularity of the approach of using computational complexity as a barrier against manipulation, this method has an important drawback: although for some voting rules the manipulation problem has been proven to be $\mathcal{NP}$-complete, these results apply only to the worst case instances; for most instances, the problem could be computationally easy. There is much evidence that this is indeed the case, including work by Friedgut et al. [7], and Isaksson et al. [9]. They prove that a single manipulator can manipulate elections with relatively high probability by simply choosing a random preference. This is true when the voting rule is far from a dictatorship, in some well-defined sense.

Additional evidence for the ease of manipulating elections on average is the work of Procaccia and Rosenschein [10], and Xia and Conitzer [12]. They connected the frequency of manipulation with the fraction of manipulators out of all the voters. Specifically, they found that for a large variety of distributions of votes, when $n = o(\sqrt{N})$, then with high probability the manipulators can affect the outcome of the elections. The opposite is true when $n = \omega(\sqrt{N})$.

The current work continues this line of research. It strengthens the results of Zuckerman et al. [14], giving an algorithm with a better approximation ratio for the Unweighted Coalitional Optimization (UCO) problem under Maximin. Equivalently, it narrows the error window of the algorithm for the decision problem CCUM under Maximin. The result can be viewed as another argument in favor of the hypothesis that most rules are usually easy to manipulate.

9. CONCLUSIONS AND FUTURE WORK

We introduced a new algorithm for approximating the UCO problem under the Maximin voting rule, and investigated its approximation guarantees. In future work, it would be interesting to prove or disprove that Algorithm 1 presented in [14] has an approximation ratio of $1\frac{2}{3}$, for those instances where there is no Condorcet winner.[2] Another direction is to implement both algorithms, so as to empirically measure and compare their performance.

Acknowledgments

We would like to thank Reshef Meir, Aviv Zohar, Jerome Lang and Noam Nisan for helpful discussions on topics of this research. We also thank Edith Elkind and Piotr Faliszewski for valuable comments on an earlier version of this paper. This work was partially supported by Israel Science Foundation grant #898/05, and Israel Ministry of Science and Technology grant #3-6797.

[2]We have an example showing that that algorithm is no better than a 2-approximation when there is a Condorcet winner.

10. REFERENCES

[1] J. Bartholdi and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8:341–354, 1991.

[2] J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241, 1989.

[3] 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.

[4] E. Elkind, P. Faliszewski, and A. Slinko. Swap bribery. In *Proceedings of SAGT 2009, Springer-Verlag LNCS 5814*, pages 299–310, October 2009.

[5] E. Elkind, P. Faliszewski, and A. Slinko. Cloning in elections. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010)*, pages 768–773, July 2010.

[6] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. In *The Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 128–133, Pasadena, California, July 2009.

[7] E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *Proc. 49th FOCS. IEEE Computer Society Press*, pages 243–249, 2008.

[8] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[9] M. Isaksson, G. Kindler, and E. Mossel. The geometry of manipulation — a quantitative proof of the Gibbard-Satterthwaite theorem. In *51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 319–328, October 2010.

[10] A. D. Procaccia and J. S. Rosenschein. Average-case tractability of manipulation in elections via the fraction of manipulators. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 718–720, Honolulu, Hawaii, May 2007.

[11] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[12] L. Xia and V. Conitzer. Generalized scoring rules and the frequency of coalitional manipulability. In *Proceedings of ACM EC-08*, pages 109–118, July 2008.

[13] L. Xia, M. Zuckerman, A. D. Procaccia, V. Conitzer, and J. S. Rosenschein. Complexity of unweighted coalitional manipulation under some common voting rules. In *The Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 348–353, Pasadena, California, July 2009.

[14] 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.

Algorithm 1 Decides CCUM for Maximin voting rule

procedure MAXIMIN(C, p, X_S, n) ▷ X_S is the set of preferences of voters in S, n is the number of voters in T
  $X \leftarrow \emptyset$ ▷ Will contain the preferences of T
  for $i = 1, \ldots, n$ **do** ▷ Iterate over voters
    $P_i \leftarrow (p)$ ▷ Put p at the first place of the i-th preference list
    Build a digraph $G^{i-1} = (V, E^{i-1})$ ▷ $V = C \setminus \{p\}$, $(x, y) \in E^{i-1}$ iff ($y \in \text{MIN}_{i-1}(x)$ and $p \notin \text{MIN}_{i-1}(x)$)
    for $c \in C \setminus \{p\}$ **do** ▷ This for loop is used in the algorithm's analysis
      if $d_{out}(c) = 0$ **then**
        $c.father \leftarrow p$
      else
        $c.father \leftarrow null$
      end if
    end for
    while $C \setminus P_i \neq \emptyset$ **do** ▷ while there are candidates to be added to i-th preference list
      Evaluate the score of each candidate based on the votes of S and $i-1$ first votes of T
      if there exists a set $A \subseteq C \setminus P_i$ with $d_{out}(a) = 0$ for each $a \in A$ **then** ▷ if there exist vertices in the digraph G^{i-1} with out-degree 0
        Add the candidates of A to the stacks Q_j, where to the same stack go candidates with the same score
        $b \leftarrow Q_1.popfront()$ ▷ Retrieve the top-most candidate from the first stack—with the lowest scores so far
        $P_i \leftarrow P_i + \{b\}$ ▷ Add b to i's preference list
      else
        Let $s \leftarrow \min_{c \in C \setminus P_i}\{S_{i-1}(c)\}$
        if there is a cycle U in G^{i-1} s.t. there are 3 vertices a, b, c, s.t. $(c, b), (b, a) \in U$, and $S_{i-1}(c) = S_{i-1}(b) = s$ **then**
          $P_i \leftarrow P_i + \{b\}$ ▷ Add b to i's preference list
        else
          Pick $b \in C \setminus P_i$ s.t. $S_{i-1}(b) = s$ ▷ Pick any candidate with the lowest score so far
          $P_i \leftarrow P_i + \{b\}$ ▷ Add b to i's preference list
        end if
      end if
      for $y \in C \setminus P_i$ **do**
        if $(y, b) \in E^{i-1}$ **then** ▷ If there is a directed edge from y to b in the digraph
          Remove all the edges of E^{i-1} originating in y
          $y.father \leftarrow b$ ▷ This statement is used in the algorithm's analysis
          Add y to the front of the appropriate stack Q_j—according to $S_{i-1}(y)$
        end if
      end for
    end while
    $X \leftarrow X \cup \{P_i\}$
  end for
  $X_T \leftarrow X$
  if argmax$_{c \in C}\{$Score of c based on $X_S \cup X_T\} = \{p\}$ **then**
    return true ▷ p wins
  else
    return false
  end if
end procedure