An Algorithm for the Coalitional Manipulation Problem under Maximin

Michael Zuckerman, Omer Lev and Jeffrey S. Rosenschein

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 $1\frac{1}{2}$ to the corresponding optimization problem. This is an improvement over a previously known algorithm that gave a 2-approximation. We also prove that our analysis is tight, i.e., there are instances on which a $1\frac{1}{2}$-approximation is the best the algorithm can achieve.

1 Introduction

Exploring the computational complexity of, and algorithms for, the *manipulation problem* is one of the most important research areas in computational social choice.

In an election, voters 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 [4, 5], 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, then there exist profiles where a voter can benefit by voting insincerely. Submitting insincere rankings in an attempt to benefit is called *manipulation*.

There are several ways to circumvent this 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 the 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 [6], 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. [7] defined a natural optimization problem for the unweighted setting (i.e., Unweighted Coalitional Optimization, UCO): finding the minimal number of manipulators that is 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 $1\frac{1}{2}$-approximation to the problem.[1] Then we provide an example showing that the approximation ratio of the

[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

algorithm is not better than $1\frac{1}{2}$.

2 The Maximin Voting Rule, Manipulation and Condorcet winner

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 2.1. 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 2.2. 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 2.3. We implicitly assume here that the manipulators have full knowledge about the non-manipulators' votes. 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 $c \in C$, $S_i(x) = \min_{y \neq x} N_i(x, y)$. Also, we denote for $x \in C$, $\mathrm{MIN}_i(x) = \{y \in C \setminus \{x\} \mid S_i(x) = N_i(x, y)\}$. We denote for $0 \leq i \leq n$, $\mathrm{ms}(i) = \max_{c \in C \setminus \{p\}} S_i(c)$. That is, $\mathrm{ms}(i)$ is the maximum score of the opponents of p after i manipulators have voted.

Definition 2.4. The *Condorcet winner* of an election is the candidate who, when compared with every other candidate, is preferred by more voters.

3 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). It works as follows: fix some order on the manipulators;

optimization problem.

the current manipulator i ranks p first. 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 iterates over the candidates that have not yet been ranked in his preference list. If there are candidates with an out-degree 0, then the manipulator adds such a candidate who has the lowest score (among the candidates with an out-degree 0) to his preference list. Note that 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 5.4 to work. Otherwise, if there are no candidates with out-degree 0, then 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 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" (this assignment is used to analyze the algorithm).

Note the subtle difference between calculating the scores in Algorithm 1 in this paper, as compared to in Algorithm 1 in [7]. 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 3.1. 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.

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, it always puts p first in the preference lists, making the score of p grow by 1 with each manipulator. Regarding p's opponents, it 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 before them). When there are no candidates without outgoing edges, the algorithm finds the candidate with the minimal score, and ranks it in the next place in the preference list. 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 from x, since if we rank x now, its score won't 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.

Definition 3.2. 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).

4 2-approximation

We first prove that Algorithm 1 has an approximation ratio of 2. We then use this result in the proof of the $1\frac{1}{2}$ approximation ratio. The proof of Theorem 4.1 via Lemma 4.2 and Lemma 4.3 is quite similar to the proof of Theorem 3.16 in [7].

Theorem 4.1. *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 one 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 4.2. *Let i be an integer, $0 \leq i \leq n-1$. 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 $S_i(c) \leq t$. 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 $c \in V^i(x)$, $S_i(c) \geq t$, and so for all $c \in V^i(x)$, $S_i(c) = t$. We claim that for all $c \in V^i(x)$, $\mathrm{MIN}_i(c) \subseteq V^i(x)$. If, by way of contradiction, there exists $c \in V^i(x)$ s.t. there is $b \in \mathrm{MIN}_i(c)$ 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 went up 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 next lemma we put forward an upper bound on the growth rate of the scores of p's opponents.

Lemma 4.3. *For all $0 \leq i \leq n-2$, $ms(i+2) \leq ms(i) + 1$*

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

Let $V^i(x)$ and $G^i(x)$ as before. By Lemma 4.2, $G^i(x)$ 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 at stage $i+1$, it follows that $S_{i+1}(b) = S_i(b) \leq t$.

Suppose, for contradiction, that $S_{i+2}(x) > t+1$. Then the score of x went up 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 $S_{i+1}(b) \leq t < t+1 = S_{i+1}(x)$)—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 $z \in Z_1$, $b \in \mathrm{MIN}_i(z)$, 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 4.1.

Proof of Theorem 4.1. Let *opt* denote the minimum size of coalition needed to make p win. It is easy to see that $opt \geq \text{ms}(0) - S_0(p) + 1$. We set $n = 2\text{ms}(0) - 2S_0(p) + 2 \leq 2opt$. Then, by Lemma 4.3:

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

Whereas:

$$S_n(p) = S_0(p) + n = 2\text{ms}(0) - S_0(p) + 2 > \text{ms}(n).$$

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

5 $1\frac{1}{2}$-approximation when there are no 2-cycles

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

Theorem 5.1. *For instances where there are no 2-cycles in the graphs G^i built by Algorithm 1, it gives a $1\frac{1}{2}$-approximation to the optimum.*

Lemma 5.2. *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 4.2. 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 5.3. *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 the stage $i + 1$, i.e., in G^{i+1}.*

Proof. In Lemma 4.2 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 went up 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 5.4. *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 to 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 5.2 and Lemma 5.3 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 an edge (w, y) with the weight t s.t. there is a path from y to w in G^{i+1}. W.l.o.g. let w be such a vertex that was picked first in the stage $i + 2$. According to the algorithm (lines 17–18), such a vertex 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 5.3) till we reach null (we cannot reach p this way since the condition 1 does not hold). Similarly to Lemma 5.3), it can be verified that the last vertex before null is y. If we passed through w this way 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 w since w is the first vertex that was picked in stage $i + 2$ s.t. there is an edge (w, y) and there is a path from y to w in G^{i+1}; and 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) 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 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 5.3, we go from a vertex to its father, starting with b, till we reach w. This way we got a path in G^{i+2} from x through y and b till w, and as mentioned earlier, $S_{i+2}(w) = t$.

□

The next lemma is central in the proof of Theorem 5.1. It states that the maximum score of p's opponents grows rather slowly.

Lemma 5.5. *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 $\mathrm{ms}(i) = t$. We need to prove that $S_{i+3}(x) \leq t + 1$. If $S_{i+1}(x) \leq t$, then similarly to Lemma 4.3 we can prove that $S_{i+3}(x) \leq t + 1$. So now we assume that $S_{i+1}(x) = t + 1$. By Lemma 4.3, 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 5.4 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 went up 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 go up 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 go up as well, and so $S_{i+3}(x) = S_{i+2} = t+1$, a contradiction. □

We are now ready to prove the main theorem.

Proof of Theorem 5.1. Let *opt* denote the minimal size of the coalition of manipulators that can make p win the election. It is easy to see that $opt \geq \text{ms}(0) - S_0(p) + 1$. We shall prove that Algorithm 1 will find a manipulation for $n = \left\lceil \frac{3\text{ms}(0) - 3S_0(p) + 3}{2} \right\rceil \leq \lceil \frac{3}{2} opt \rceil$. And indeed, by Lemma 5.5,

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

Whereas,

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

□

6 The approximation ratio when there are 2-cycles

Theorem 6.1. *The $1\frac{1}{2}$-approximation ratio of Algorithm 1 is valid also when there are 2-cycles in the graphs built by the algorithm.*

Proof. This following proof will show, in a way, that our algorithm is optimal in dismantling 2-cycles—if there are 2-cycles in G^i, then for every algorithm $ms^*(i) \geq ms(i)$. Once 2-cycles have been dismantled (and they cannot return), our algorithm performs a $1\frac{1}{2}$-approximation on the number of steps left, and thus, generally a $1\frac{1}{2}$-approximation on the optimal solution.

Several lemmas will lead us to this proof:

Lemma 6.2. *If there are no cycles of length 2 in a certain stage of the algorithm run (G^i), then no 2-cycles will be created in any further iteration—G^j for $j > i$ will have no 2-cycles.*

Proof. We shall prove it by reverse induction, showing that if there is a cycle of length 2 in G^j, there is also one in G^{j-1}. Suppose G^j contains $a, b \in C$ such that $(a,b), (b,a) \in E^j$, and $S_j(a) \geq S_j(b)$. Thus, $S_j(a) + S_j(b) = N + j$. Suppose for contradiction there is no 2-cycle in G^{j-1}, and suppose that $S_j(a) = S_{j-1}(a)$, which means $N_{j-1}(a,b) = N_j(a,b) = S_j(a)$.

Hence b was selected before a, and so $N_j(b,a) = N_{j-1}(b,a) + 1$. But since we're assuming the cycle didn't exist in G^{j-1}, $N_{j-1}(b,a) > S_{j-1}(b)$ - and thus $N_j(b,a) = N_{j-1}(b,a) + 1 > S_{j-1}(b) + 1 \geq S_j(b)$, contradicting $S_j(b) = N_j(b,a)$.

If $S_j(a) = S_{j-1}(a) + 1$: this means that $S_{j-1}(a) = S_{j-1}(b)$, otherwise a would not increase its score. If $S_{j-1}(b) = S_j(b)$, then $S_{j-1}(b) = N_{j-1}(b,a) = N_j(b,a)$, and hence a was selected before b, and it means $N_{j-1}(a,b) = S_{j-1}(a)$ - there was a 2-cycle in G^{j-1}. If $S_j(b) = S_j(a) = S_{j-1}(b) + 1$ - both a and b increased their score. Let's take a look at the set of all vertices with maximal score in G^{j-1}: $D = \{c \in C | S_{j-1}(c) = S_{j-1}(a)\}$. For every $e, e' \in D$, either $(e,e') \in G^{j-1}$ or $(e',e) \in G^{j-1}$ (or both - but that would mean the existence of a 2-cycle). Due to the algorithm's selection process (lines 21-22 of the algorithm), only one vertex will increase its score in this set, and it can't be both a and b. □

Lemma 6.3. *If $c \in C$ is not a part of a 2-cycle in G^i and is a part of a 2-cycle in G^j for some $j > i$, then there are no 2-cycles in G^{j+2}.*

Proof. Suppose j is the minimal index for which c is part of a 2-cycle, and suppose its partner for the 2-cycle is a. According to Lemma 6.2, there must have been a 2-cycle in G^{j-1} which involved a with another vertex $b \neq c$. (Otherwise, if neither a nor c were in a 2-cycle - suppose d and e were and $S_{j-1}(d) \geq S_{j-1}(e)$ - hence $S_{j-1}(a) \leq N_{j-1}(e,d) + 1$ and similarly for $S_{j-1}(c)$. Thus, $S_j(a), S_j(c) \leq N_{j-1}(e,d)$, but since $S_j(a) + S_j(c) = N + j$, we reach a contradiction). If $S_{j-1}(a) = S_j(a)$, then $N_j(a,c) = N_{j-1}(a,c) = S_{j-1}(a)$, and hence $N_j(c,a) = N_{j-1}(c,a) + 1$, but since $S_j(c) = S_{j-1}(c) + 1$ (otherwise, there would be no 2-cycle in G^j with c), $N_{j-1}(c,a) = S_{j-1}(c)$ - and a and c formed a 2-cycle in G^{j-1}, contradicting our assumption.

If $S_{j-1}(a) + 1 = S_j(a)$, this means that $S_{j-1}(a) = S_{j-1}(b)$ (otherwise, a would be a Condorcet winner, and its score wouldn't increase), and since a and b formed a 2-cycle in stage $j-1$, while a and c form one in stage j, $S_j(b) = S_{j-1}(b) = S_j(c)$, and $N_j(b,c) = S_j(b)$ or $N_j(c,b) = S_j(c)$ (we shall assume w.l.o.g. the latter). Due to lines 21-22 of the algorithm, b should be selected before c - if not, this means $d_{out}(c) = 0$, - and therefore $S_{j+1}(c) = S_j(c)$. $S_{j+1}(a) = S_j(a)$ as well, as either it is selected after b and c or when $d_{out}(a) = 0$. Therefore, the cycle between a and c is broken, and at most, a cycle is left between a and b, with $S_{j+1}(a) = S_{j+1}(b)$, and $N_{j+1}(a,c) = S_{j+1}(a)$. Moving on to stage $j+2$, if c is chosen before a and b, then $d_{out}(a) = 0$, and hence $S_{j+2}(a) = S_{j+1}(a)$ and $S_{j+2}(b) = S_{j+1}(b)$, and there is no cycle. If c was chosen after either b or a, it must be since one of them has $d_{out} = 0$, and thus, as in the previous case, $S_{j+2}(a) = S_{j+1}(a)$ and $S_{j+2}(b) = S_{j+1}(b)$.

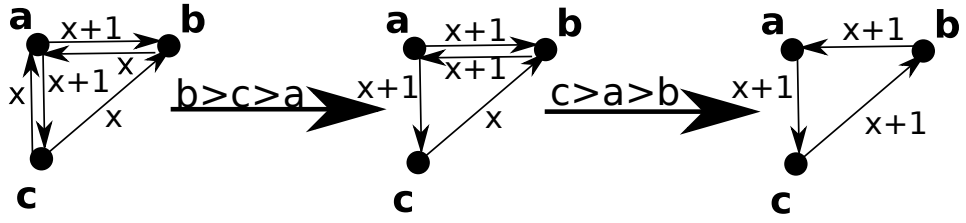

Figure 1: The case of $S_j(a) = S_{j-1}(a) + 1$

□

Lemma 6.4. *If there is more than one 2-cycle in G^0, there will either be no 2-cycles in G^3 or, there will be no 2-cycles after j stages and $ms(j) = ms(0)$.*

Proof. There are several cases which need to be considered here:

- If there is no Condorcet winner - This means there is a set of vertices - $a, b_1, \ldots, b_r$ that form a complete graph between them, and $S_0(a) = S_0(b_i) = \frac{N}{2} = x$. After the first step, there will be one vertex - a (w.l.o.g.) for which $S_1(a) = x + 1$, and for the rest $S_1(b_i) = N_1(b_i, a) = x$, and for each b_i,b_j either $N_1(b_i, b_j) = x$ or $N_1(b_j, b_i) = x$. From here, the 2-cycles are destroyed in the same manner of the end of the previous lemma - which takes 2 steps, making G^3 2-cycle free.

- If a is a Condorcet winner, forming 2-cycles with $b_1, \ldots, b_r$ and $S_0(b_i) = S_0(a) - 1$. This is the same state reached by the algorithm after one step in the previous item, and hence there will be no 2-cycles in G^2.

- If a is a Condorcet winner, forming 2-cycles with $b, c_1, \ldots, c_r$, and $S_0(a) > S_0(b) + 1 = S_0(c_i) + 1$. In the first step, if a is selected first, it is since $d_{out}(a) = 0$, and there are no more any 2-cycles. Suppose b (w.l.o.g.) was selected before all others - this made $d_{out}(a) = 0$, and hence $S_1(b) = S_0(b) + 1$, $S_1(a) = S_0(a)$ and $S_1(c_i) = S_0(c_i)$. From now on, as shown in Lemma 6.3, the only 2 vertices in a 2-cycle will be a and b. If a or b are ever selected, at any stage, before any of the c_i, this means that both their scores remained the same, and hence the 2-cycle is broken. If there is a c_ℓ for which $N_1(c_\ell, b) = S_0(b)$, the 2-cycle will be dismantled as can be seen in the figure below. As long as the 2-cycle exists, $N_k(b, c_i) = N_1(b, c_i)$, and also, since we know $N_1(c_i, b) \geq S_0(c_i) = S_0(b)$, $N_k(b, c_i) \leq S_0(a)$, and after $N_1(b, c_i) - S_0(b)$ steps, $S_{N_1(b,c_i)-S_0(b)}(b) = N_1(b, c_i) = N_{N_1(b,c_i)-S_0(b)}(b, c_i) = N_{N_1(b,c_i)-S_0(b)}(b, a)$, and after the next step, the 2-cycle will be broken.

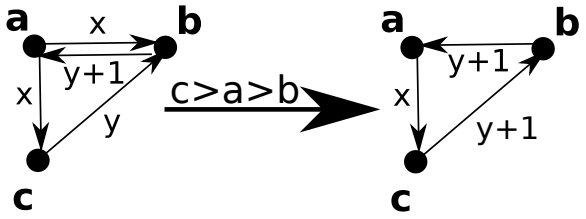

Figure 2: Dismantling multiple 2-cycles

□

Therefore, we can assume a and b were part of a 2-cycle in G^0, and they will be the only participants of a 2-cycle during the algorithm's run.

To continue we need a few definitions. We'll define $c \in C$ as $c \in \{x \in C | x \in \min_{y \in C \setminus \{a,b\}} N_0(b, y)\}$. We'll define $d \in C$ as $d \in \{x \in C | x \in \min_{y \in C \setminus \{a,b\}} N_0(a, y)\}$.

Lemma 6.5. *If $N_0(b, c) < S_0(a)$, then after $N_0(b, c) - S_0(b) + 1$ steps there are no 2-cycles, and $ms(N_0(b, c) - S_0(b) + 1) = ms(0)$.*

Proof. To simplify notation, we shall denote $w = N_0(b, c) - S_0(b)$.

$S_i(a) = S_0(a)$ for $i \leq w + 1$ (since a is still the Condorcet winner). We deal with the case when $S_w(b) = N_w(b, a)$ - otherwise we don't have a 2-cycle at all. In stage w, $S_w(b) = N_w(b, a) = S_0(b) + N_0(b, c) - S_0(b) = N_0(b, c)$. If c was ever selected after b, it was if at some stage $j < w$, $d_{out}(b) = 0$, and then $S_j(b) = S_{j-1}(b)$ - but since $S_j(a) = S_{j-1}(a)$, this means the 2-cycle was broken at that stage, so we may assume $S_w(b) = N_w(b, c)$. Since $S_w(b) > S_w(c)$ (otherwise there would be a 2-cycle with a and c), if b is selected before c, this means $d_{out}(b) = 0$, and $S_w(b) = S_{w+1}(b)$, otherwise, c is selected before b, and

$S_{w+1}(b) = N_{w+1}(b,c) = S_w(b)$. Since both $S_{w+1}(a) = S_w(a)$ and $S_{w+1}(b) = S_w(b)$, there is no 2-cycle between a and b. □

Lemma 6.6. *Let $h = min(N_0(b,c), N_0(a,d))$. For any algorithm, if $h \geq S_0(a)$, $ms^*(S_0(a) - S_0(b) + 2(h - S_0(a))) \geq h$.*

Proof. To simplify notation, we shall denote $w = S_0(a) - S_0(b) + 2(h - S_0(a))$.

Suppose there is an algorithm for which, in the w step, $ms^*(w) < h$. This means that $S^*_w(a), S^*_w(b) < h$, so a and b aren't part of a 2-cycle in G^{*w} (if they were, $S^*_w(a) + S^*_w(b) = N + w = N - N + 2h = 2h$, so one of them would have to be $\geq h$). There are c', d' such that $N_w(b,c') < h$ and $N_w(a,d') < h$. But since $N_w(b,c') \geq N_0(b,c')$ and $N_w(a,d') \geq N_0(a,d')$, we reach a contradiction to the definition of h. □

Lemma 6.7. *Let $h = min(N_0(b,c), N_0(a,d))$. Using Algorithm 1, if $h \geq S_0(a)$, $ms(S_0(a) - S_0(b) + 2(h - S_0(a))) = h$, and there are no 2-cycles in $G^{S_0(a) - S_0(b) + 2(h - S_0(a)) + 2}$.*

Proof. To simplify notation, we shall denote $w = S_0(a) - S_0(b) + 2(h - S_0(a))$.

As long as a is the Condorcet winner (i.e, $i \leq S_0(a) - S_0(b)$), $ms(i) = S_0(a)$, since a's score will not increase. For the next $2(h - S_0(a))$ steps, a and b will remain in a 2-cycle - $N_j(a,b) < h$ and $N_j(b,a) < h$, and there is no other vertex e for which $N_j(a,e) < h$ or $N_j(b,e) < h$. Furthermore, for $j < w$, $N_j(a,d) = N_0(a,d)$, and $N_j(b,c) = N_0(b,c)$. This, since $S_j(a) \geq S_j(b) \geq S_j(x)$ for $x \in C \setminus \{a,b\}$ (due to a and b being Condorcet winners), if c was selected after b, it was since $d_{out}(b) = 0$ - but that would require that $S_j(b) \geq h$ (and similarly for d being selected after a).

In G^w, suppose, w.l.o.g, that a is the Condorcet winner (it can alternate between a and b). In this stage, either $(a,d) \in E^w$, or $(b,c) \in E^w$. If $(a,d) \in E^w$, this is the case dealt with in Lemma 6.3. If $(b,c) \in E^w$, then c will be selected before b, or b was selected when $d_{out}(b) = 0$ - so $S_{w+1}(b) = S_w(b)$. Similarly, a's score won't change - either b was selected before it, or $d_{out}(a) = 0$ - so $S_{w+1}(a) = S_w(a)$, and the 2-cycle is abolished. □

We have shown that if there are multiple 2-cycles in G^0, we end up with no 2-cycles in G^3. If there is one, it is abolished, and the ms at the state in which it is abolished is the smallest possible. □

7 Lower bound on the approximation ratio of the algorithm

Now we show that our analysis of Algorithm 1 is accurate.

Theorem 7.1. *The $1\frac{1}{2}$ approximation ratio of Algorithm 1 is asymptotically tight.*

Proof. Consider the following example (see Figure 3). $C = \{p, a_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_l, b_l, c_l\}$. Let k be an integer, $\frac{N}{3} \leq k < \frac{N}{2}$. $S_0(p) = 0$; for all $j, 1 \leq j \leq l$: $S_0(a_j) = N_0(a_j, b_j) = S_0(b_j) = N_0(b_j, c_j) = S_0(c_j) = N_0(c_j, a_j) = k$. In addition, for each $j, 1 \leq j \leq l-1$: $N_0(a_j, a_{j+1}) = k+1$, and $N_0(a_l, a_1) = k+1$. When showing the preferences of the manipulators, 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 $c_j \succ b_j \succ a_j$. Consider the following preference list of the manipulators:

$$p \succ A_l \succ A_{l-1} \succ \ldots \succ A_1$$
$$p \succ A_{l-1} \succ A_{l-2} \succ \ldots \succ A_1 \succ A_l$$
$$p \succ A_{l-2} \succ A_{l-3} \succ \ldots \succ A_1 \succ A_l \succ A_{l-1}$$
$$\ldots$$

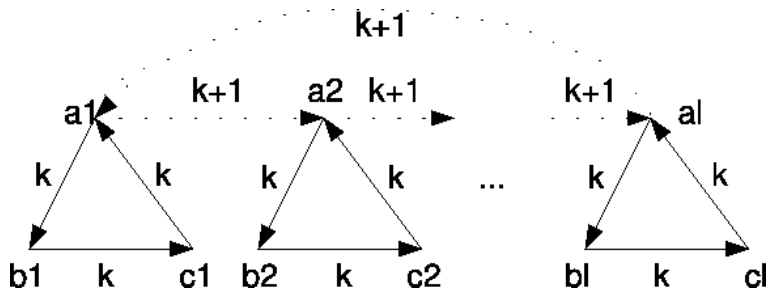

Figure 3: Example

It can be verified that in the above preference list, the maximum score of p's opponents ($\text{ms}(i)$) 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:

$$\begin{aligned}&p \succ C_1 \succ C_2 \succ \ldots \succ C_l \\ &p \succ B_2 \succ B_3 \succ \ldots \succ B_l \succ B_1 \\ &p \succ A_3 \succ A_4 \succ \ldots \succ A_l \succ A_1 \succ A_2 \\ &p \succ C_4 \succ C_5 \succ \ldots \succ C_l \succ C_1 \succ C_2 \succ C_3 \\ &\ldots\end{aligned}$$

It can be verified that in the above preference list, $\text{ms}(i)$ 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:

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

We then 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 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 [7] has an approximation ratio of $1\frac{1}{2}$, for those instances where there is no Condorcet winner.[2] Another issue is to implement both algorithms, to empirically measure their performance and compare them in practice.

Acknowledgments

We would like to thank Reshef Meir, Aviv Zohar, and Jerome Lang for helpful discussions on the topics of this work. This work was partially supported by Israel Science Foundation grant #898/05, and Israel Ministry of Science and Technology grant #3-6797.

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

[5] 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.

[6] Lirong Xia, Michael Zuckerman, Ariel D. Procaccia, Vincent Conitzer, and Jeffrey 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.

[7] Michael Zuckerman, Ariel D. Procaccia, and Jeffrey S. Rosenschein. Algorithms for the coalitional manipulation problem. *Journal of Artificial Intelligence*, 173(2):392–412, February 2009.

Michael Zuckerman, Omer Lev, Jeffrey S. Rosenschein
School of Engineering and Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
Email: michez@cs.huji.ac.il
Email: omerl@cs.huji.ac.il
Email: jeff@cs.huji.ac.il

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

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 ($y \in \mathrm{MIN}_{i-1}(x)$ and $p \notin \mathrm{MIN}_{i-1}(x)$)
    for $c \in C \setminus \{p\}$ **do** ▷ This for loop is used in the 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 algorithm 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 $\mathrm{argmax}_{c \in C}\{\text{Score of } c \text{ based on } X_S \cup X_T\} = \{p\}$ **then**
    return true ▷ p wins
  else
    return false
  end if
end procedure