

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

¹Strictly 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 $C = \{c_1, \dots, c_m\}$ of candidates, and a set $S = \{v_1, \dots, v_{|S|}\}$ 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 $N_i(x, y) = |\{j \mid x \succ_j y, \succ_{j \in S \cup \{1, \dots, i\}}\}|$. 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 $x \in C$, $\text{MIN}_i(x) = \{y \in C \setminus \{x\} \mid S_i(x) = N_i(x, y)\}$. We denote for $0 \leq i \leq n$, $\text{ms}(i) = \max_{c \in C \setminus \{p\}} S_i(c)$. That is, $\text{ms}(i)$ 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 $(y \in \text{MIN}_{i-1}(x) \text{ and } p \notin \text{MIN}_{i-1}(x))$. 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 5.5 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 $G_x^i = (V_x^i, E_x^i)$ be the graph G^i reduced to the vertices that were ranked below x in stage $i+1$, including x . Let $V^i(x) = \{y \in V_x^i \mid \text{there is a path in } G_x^i \text{ from } x \text{ to } y\}$. Also, let $G^i(x)$ be the sub-graph of G_x^i 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 $c \in V^i(x)$, $\text{MIN}_i(c) \subseteq V^i(x)$. 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 $b \notin V_x^i$, since otherwise, if $b \in V_x^i$, then from $c \in V^i(x)$ and $(c, b) \in E_x^i$ we get that $b \in V^i(x)$. So $b \notin V_x^i$, 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 $c \in V^i(x) \subseteq V_x^i$. 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). \square

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 $t = ms(i)$. 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 $z \in Z_1$, $b \in \text{MIN}_i(z)$, 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, \dots , 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. \square

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 $opt \geq ms(0) - S_0(p) + 1$. We set $n = 2ms(0) - 2S_0(p) + 2 \leq 2opt$. Then, by Lemma 4.3:

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

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 . \square

5 $1\frac{1}{2}$ -approximation

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.*

We first prove the following lemma regarding the length of the cycles in the digraphs built by the algorithm.

Lemma 5.2. *If for all $c \in C \setminus \{p\}$ it holds that $S_0(c) < \lfloor \frac{N}{2} \rfloor$, then during the run of the entire algorithm, in the graph built by the algorithm, there will be no cycles of length 2.*

Proof. Suppose that for all $c \in C \setminus \{p\}$ it holds that $S_0(c) < \lfloor \frac{N}{2} \rfloor$. By Lemma 4.3, it holds for all $c \in C \setminus \{p\}$ and all $0 \leq i \leq n - 2$, that $S_{i+2}(c) \leq ms(i) + 1$. Then for all $0 \leq i \leq n$:

$$S_i(c) \leq ms(0) + \left\lceil \frac{i}{2} \right\rceil < \left\lfloor \frac{N}{2} \right\rfloor + \left\lceil \frac{i}{2} \right\rceil \leq \left\lfloor \frac{N+i}{2} \right\rfloor.$$

Now if, by way of contradiction, there is a cycle of length 2 between vertices x and y after stage i , then $S_i(x) = N_i(x, y) < \left\lfloor \frac{N+i}{2} \right\rfloor$ and $S_i(y) = N_i(y, x) < \left\lfloor \frac{N+i}{2} \right\rfloor$, and then $S_i(y) = N_i(y, x) \leq \left\lfloor \frac{N+i}{2} \right\rfloor$. Hence, $N + i = N_i(x, y) + N_i(y, x) < \left\lfloor \frac{N+i}{2} \right\rfloor + \left\lfloor \frac{N+i}{2} \right\rfloor = N + i$, a contradiction. \square

Lemma 5.3. *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 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 $i + 1$. \square

Lemma 5.4. *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 $z_2 = z_1.\text{father}$. As observed earlier, $z_2 \neq \text{null}$. 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 $z_2 = z_1.\text{father}$. Therefore, the “father” field of z_2 after stage $i + 1$ is not null. Let $z_3 = z_2.\text{father}$. If $z_3 = x$ then we are done because we have found a cycle $x \rightarrow z_1 \rightarrow z_2 \rightarrow z_3 = x$ 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}, \dots, 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 $x \rightarrow z_1 \rightarrow \dots \rightarrow z_{j-1} \rightarrow x$ which was ranked in the reverse order in stage $i + 1$. \square

Lemma 5.5. *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 .*

The proof of this lemma uses the same ideas as the proof of Lemma 5.4, and is omitted due to space limitations.

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 5.6. *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 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 5.5 there exists a vertex w s.t. there is a path in G^{i+2} from x to w , and $p \in MIN_{i+2}(w)$ 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 $S_{i+3}(x) = S_{i+2}(x) = t + 1$, a contradiction. \square

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 $opt \geq ms(0) - S_0(p) + 1$. We shall prove that Algorithm 1 will find a manipulation for $n = \left\lceil \frac{3ms(0) - 3S_0(p) + 3}{2} \right\rceil \leq \left\lceil \frac{3}{2}opt \right\rceil$. And indeed, by Lemma 5.6,

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

Whereas,

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

□

Theorem 5.7. *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. Due to space constraints, we will only provide a sketch of the proof, and we will omit the proofs of the lemmas (except of Lemma 5.9).

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.

Lemma 5.8. *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 ($j > i$) will have no 2-cycles.*

From now on, we shall assume G^0 contains at least one 2-cycle, with a Condorcet winner a .

Lemma 5.9. *If there is more than one 2-cycle at any stage i , there are no 2-cycles at stage $i + 3$.*

Proof. Suppose b_1, b_2, \dots, b_t are the 2-cycle partners of a . Suppose $S_i(a) = N_i(a, b_k) = x$ and $S_i(b_k) = N_i(b_k, a) = y$.

If $x = y$, then each $N_i(b_k, b_r) = x$. In stage $i + 1$, one vertex (w.l.o.g., a) will have a score of $x + 1$, and the rest have a score of x . This is the same situation as before (multiple 2-cycles with a), but with $x + 1 \neq x$. Now we will show that if $x \neq y$ we can eliminate 2-cycles in 2 stages (so if $x = y$, we need a total of 3 stages).

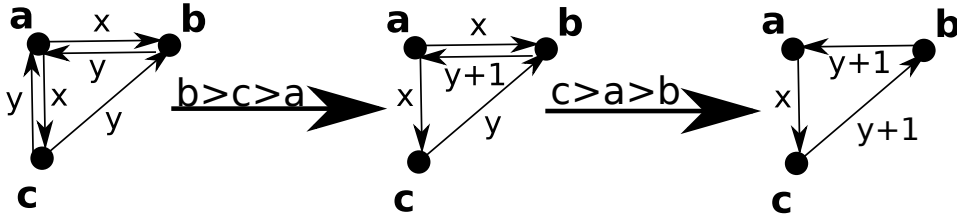


Figure 1: Dismantling multiple 2-cycles

Between each b_r, b_k there is at least one edge with value y . At stage $i + 1$, $S_{i+1}(a) = x$ (as it is the Condorcet winner), some b 's will have a value of $y + 1$, while some will have a value of y . Of those with the value y , there will be at least one, b_ℓ , for which $N_{i+1}(a, b_\ell) = x$. This is because some b_k will be selected before a , either because $d_{out}(b_k) = 0$ or because it was according to lines 21–22 of the algorithm, which ensure that at least one b_k score will not change, and it will be selected before a according to line 17. In stage $i + 2$, $S_{i+2}(a) = x$, since either b_ℓ will be selected before a , or if not, this means a was selected when $d_{out}(a) = 0$, which occurs when a 's value doesn't change. Furthermore, if $S_{i+1}(b_r) = y + 1$, and b_r was selected before b_ℓ , this means it happened due to $d_{out}(b_r) = 0$, and thus $S_{i+2}(b_r) = y + 1$, and if it was selected after b_ℓ , since a 's d_{out} after b_ℓ 's selection is 0, if b_r is selected it is either before a , when its $d_{out} = 0$, or after a , and thus $S_{i+2}(b_r) = y + 1$. Since $S_{i+2}(a) = x$, and there is no $c \in C$ such that $S_{i+2} = y + 2$, this means there are no 2-cycles in G^{i+2} . □

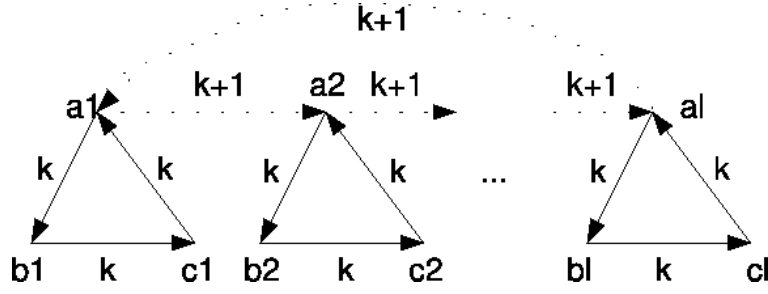


Figure 2: Example

Thus, if there is more than one 2-cycle in G^0 , it will be eliminated in 3 steps at the most. Now, we wish to show that the algorithm eliminates 2-cycles as fast as possible, and then maintains its $1\frac{1}{2}$ -approximation. Once the 2-cycles are eliminated the ms is the same (or less) than the optimal algorithm, and from that step onward our algorithm gives a $1\frac{1}{2}$ -approximation.

Lemma 5.10. *Let b be a 's partner in the 2-cycle in stage G^i . If in G^i there is $c \in C \neq a, b$ such that $N_i(a, c) = S_i(a)$, there will be no 2-cycle in G^{i+3} .*

Lemma 5.11. *If $c \in C$ was not a part of a 2-cycle in G^i , and is a part of a 2-cycle in G^{i+1} , there will be no 2-cycles in G^{i+4} .*

Therefore, we can assume that 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 will define $c \in C$ as $c \in \{x \in C \mid x \in \min_{y \in C \setminus \{a, b\}} N_0(b, y)\}$. We will define $d \in C$ as $d \in \{x \in C \mid x \in \min_{y \in C \setminus \{a, b\}} N_0(a, y)\}$.

Lemma 5.12. *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 $ms(N_0(b, c) - S_0(b) + 1) = ms(0)$.*

Lemma 5.13. *Let $h = \min(N_0(b, c), N_0(a, d))$. For any algorithm, if $h \geq S_0(a)$, $ms^*(S_0(a) - S_0(b) + 2(h - S_0(a))) \geq h$.*

Lemma 5.14. *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 $G^{S_0(a) + 2(h - S_0(a)) + 1}$.*

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. From that point on, our algorithm provides a $1\frac{1}{2}$ -approximation (according to Theorem 5.1). \square

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

Theorem 5.15. *The $1\frac{1}{2}$ approximation ratio of Algorithm 1 is asymptotically tight.*

Proof. Consider the following example (see Figure 2). $C = \{p, a_1, b_1, c_1, a_2, b_2, c_2, \dots, a_l, b_l, c_l\}$. 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:

$$\begin{aligned} p &\succ A_l \succ A_{l-1} \succ \dots \succ A_1 \\ p &\succ A_{l-1} \succ A_{l-2} \succ \dots \succ A_1 \succ A_l \\ p &\succ A_{l-2} \succ A_{l-3} \succ \dots \succ A_1 \succ A_l \succ A_{l-1} \\ &\dots \end{aligned}$$

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:

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

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:

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

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. \square

6 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: omer1@cs.huji.ac.il
Email: jeff@cs.huji.ac.il

²We 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

```
1: procedure MAXIMIN( $C, p, X_S, n$ )  $\triangleright X_S$  is the set of preferences of voters in  $S$ ,  $n$  is the
   number of voters in  $T$ 
2:    $X \leftarrow \emptyset$   $\triangleright$  Will contain the preferences of  $T$ 
3:   for  $i = 1, \dots, n$  do  $\triangleright$  Iterate over voters
4:      $P_i \leftarrow (p)$   $\triangleright$  Put  $p$  at the first place of the  $i$ -th preference list
5:     Build a digraph  $G^{i-1} = (V, E^{i-1})$   $\triangleright V = C \setminus \{p\}, (x, y) \in E^{i-1}$  iff
   ( $y \in \text{MIN}_{i-1}(x)$  and  $p \notin \text{MIN}_{i-1}(x)$ )
6:     for  $c \in C \setminus \{p\}$  do  $\triangleright$  This for loop is used in the analysis
7:       if  $d_{out}(c) = 0$  then
8:          $c.father \leftarrow p$ 
9:       else
10:         $c.father \leftarrow null$ 
11:      end if
12:    end for
13:    while  $C \setminus P_i \neq \emptyset$  do  $\triangleright$  while there are candidates to be added to  $i$ -th preference
   list
14:      Evaluate the score of each candidate based on the votes of  $S$  and  $i - 1$  first
   votes of  $T$ 
15:      if there exists a set  $A \subseteq C \setminus P_i$  with  $d_{out}(a) = 0$  for each  $a \in A$  then  $\triangleright$  if
   there exist vertices in the digraph  $G^{i-1}$  with out-degree 0
16:        Add the candidates of  $A$  to the stacks  $Q_j$ , where to the same stack go
   candidates with the same score
17:         $b \leftarrow Q_1.popfront()$   $\triangleright$  Retrieve the top-most candidate from the first
   stack—with the lowest scores so far
18:         $P_i \leftarrow P_i + \{b\}$   $\triangleright$  Add  $b$  to  $i$ 's preference list
19:      else
20:        Let  $s = \min_{c \in C \setminus P_i} \{S_{i-1}(c)\}$ 
21:        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
22:           $P_i \leftarrow P_i + \{b\}$   $\triangleright$  Add  $b$  to  $i$ 's preference list
23:        else
24:          Pick  $b \in C \setminus P_i$  s.t.  $S_{i-1}(b) = s$   $\triangleright$  Pick any candidate with the lowest
   score so far
25:           $P_i \leftarrow P_i + \{b\}$   $\triangleright$  Add  $b$  to  $i$ 's preference list
26:        end if
27:      end if
28:      for  $y \in C \setminus P_i$  do
29:        if  $(y, b) \in E^{i-1}$  then  $\triangleright$  If there is a directed edge from  $y$  to  $b$  in the
   digraph
30:          Remove all the edges of  $E^{i-1}$  originating in  $y$ 
31:           $y.father \leftarrow b$   $\triangleright$  This statement is used in algorithm analysis
32:          Add  $y$  to the front of the appropriate stack  $Q_j$ —according to  $S_{i-1}(y)$ 
33:        end if
34:      end for
35:    end while
36:     $X \leftarrow X \cup \{P_i\}$ 
37:  end for
38:   $X_T \leftarrow X$ 
39:  if  $\text{argmax}_{c \in C} \{\text{Score of } c \text{ based on } X_S \cup X_T\} = \{p\}$  then
40:    return true  $\triangleright p$  wins
41:  else
42:    return false
43:  end if
44: end procedure
```
