Fine-Grained Complexity and Algorithms for the Schulze Voting Method

KRZYSZTOF SORNAT, Massachusetts Institute of Technology, USA
VIRGINIA VASSILEVSKA WILLIAMS, Massachusetts Institute of Technology, USA
YINZHAN XU, Massachusetts Institute of Technology, USA

We study computational aspects of a well-known single-winner voting rule called the Schulze method [Schulze, 2003] which is used broadly in practice. In this method the voters give (weak) ordinal preference ballots which are used to define the weighted majority graph of direct comparisons between pairs of candidates. The choice of the winner comes from indirect comparisons in the graph, and more specifically from considering directed paths instead of direct comparisons between candidates.

When the input is the weighted majority graph, to our knowledge, the fastest algorithm for computing all winners in the Schulze method uses a folklore reduction to the All-Pairs Bottleneck Paths (APBP) problem and runs in $O(m^{2.69})$ time, where m is the number of candidates. It is an interesting open question whether this can be improved. Our first result is a combinatorial algorithm with a nearly quadratic running time for computing all winners. This running time is essentially optimal as it is nearly linear in the size of the weighted majority graph.

If the input to the Schulze winners problem is not the weighted majority graph but the preference profile, then constructing the weighted majority graph is a bottleneck that increases the running time significantly; in the special case when there are m candidates and $n = O(m)$ voters, the running time is $O(m^{2.69})$, or $O(m^{2.5})$ if there is a nearly-linear time algorithm for multiplying dense square matrices.

To address this bottleneck, we prove a formal equivalence between the well-studied Dominance Product problem and the problem of computing the weighted majority graph. As the Dominance Product problem is believed to require at least time $r^{2.5-o(1)}$ on $r \times r$ matrices, our equivalence implies that constructing the weighted majority graph in $O(m^{2.499})$ time for m candidates and $n = O(m)$ voters would imply a breakthrough in the study of "intermediate" problems [Lincoln et al., 2020] in fine-grained complexity. We prove a similar connection between the so called Dominating Pairs problem and the problem of verifying whether a given candidate is a winner.

Our paper is the first to bring fine-grained complexity into the field of computational social choice. Previous approaches say nothing about lower bounds for problems that already have polynomial time algorithms. By bringing fine-grained complexity into the picture we can identify voting protocols that are unlikely to be practical for large numbers of candidates and/or voters, as their complexity is likely, say at least cubic.

CCS Concepts: • **Theory of computation → Algorithmic game theory and mechanism design**; *Graph algorithms analysis*; • **Applied computing** → *Voting / election technologies.*

Additional Key Words and Phrases: voting, Schulze method, algorithms, fine-grained complexity

ACM Reference Format:
Krzysztof Sornat, Virginia Vassilevska Williams, and Yinzhan Xu. 2021. Fine-Grained Complexity and Algorithms for the Schulze Voting Method. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC '21), July 18–23, 2021, Budapest, Hungary.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3465456.3467630

This work is licensed under a Creative Commons Attribution International 4.0 License.

EC '21, July 18–23, 2021, Budapest, Hungary.
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8554-1/21/07.
https://doi.org/10.1145/3465456.3467630

1 INTRODUCTION

We study computational aspects of *the Schulze method*[1], a single-winner voting rule defined on (weak) ordinal preference ballots. The method was introduced by Markus Schulze [39, 40] and has been extensively researched over the years—see, e.g., a survey by Schulze [41].

The Schulze voting method definition uses a representation of the votes called *the weighted majority graph*, a graph whose vertices are the candidates and whose directed weighted edges roughly capture all pairwise comparisons between candidates (see a formal definition in Section 2). The Schulze method computes, in the weighted majority graph, for every pair of candidates u and v the smallest weight $B(u, v)$ of an edge on *a widest path* between u and v. Here a widest path is a u-v path whose minimum edge weight is maximized over all u-v paths. Then, a *Schulze winner* is any candidate u s.t. for all other candidates v, $B(u, v) \geq B(v, u)$. A Schulze winner always exists [40], though it may not be unique. The method can be used to provide a (weak) order over alternatives, and hence can be seen as a preference aggregation method or multi-winner voting rule (taking the top candidates in the ranking as winners).

Schulze defined this method by modifying *the Minimax Condorcet method*[2] to address several criticisms of the Minimax Condorcet method [38, 44, 46]. Schulze's modified method is *Condorcet consistent*, i.e., a candidate who is preferred by a majority over every other candidate in pairwise comparisons (a Condorcet winner) is a winner in the Schulze method. Schulze's modification results in satisfying a desired set of axiomatic properties not satisfied by Minimax Condorcet:

- *clone independence:* This criterion proposed by Tideman [46] requires that a single-winner voting rule should be independent of introducing similar candidates (see also, e.g., [5, 11, 18]).
- *reversal symmetry:* This criterion proposed by Saari [38] means that if candidate v is the unique winner, then v must not be a winner in an election with all the preference orders inverted.
- *Smith criterion:* The Smith set is the smallest non-empty subset of candidates who (strictly) win pairwise comparison with every candidate not in the subset. The Smith criterion is satisfied when the set of winners is a subset of the Smith set [44].

The Schulze method is a voting rule used broadly in many organizations, e.g., the Wikimedia Foundation[3], Debian Project[4] and, e.g, in the software *LiquidFeedback* [5] (see also a comprehensive list of users in [41, Section 1]).

In this paper we focus on computational aspects of the Schulze method. In particular: 1) constructing a weighted majority graph; 2) checking whether a particular candidate is a winner; 3) finding a winner; 4) listing all winners.

Indicating a winner or a set of all winners is the main goal of a voting rule (tasks 3 and 4 above). The definition of the Schulze method consists of two steps that can be studied separately—task 1 above is the first of the two steps; it is useful for other voting rules as well. The decision version of the problem (task 2) is useful when considering the computational complexity of the problem [12], or when considering bribery, manipulation and control issues [24, 25, 34, 35] where one needs to verify whether a particular candidate would become a winner or non-winner.

Given a weighted majority graph, the most general task is to create a (weak) order over the alternatives (as a preference aggregation method). The best known algorithm for this task, and also

[1] The method appears under different names such as *Beatpath Method/Winner*, *Path Voting*, *(Cloneproof) Schwartz Sequential Dropping*—see discussion in [39].

[2] Also referred to as *the Minimax method* and *the Simpson-Kramer method* [40].

[3] *The Wikimedia Foundation elections to the Board of Trustees (2011)*, https://meta.wikimedia.org/wiki/Wikimedia_Foundation_elections/Board_elections/2011/en, [Online; accessed 26-May-2021].

[4] *Constitution for the Debian Project (v1.7)*, https://www.debian.org/devel/constitution.en.html, [Online; accessed 26-May-2021].

for tasks 3 and 4 above, has a running time of $O(m^{2.69})$ in terms of the number of candidates m (see, e.g., [50]). This algorithm relies on fast matrix multiplication techniques that can be impractical. The fastest algorithm without fast matrix multiplication (i.e. combinatorial) has cubic running time [39]. While the running time is polynomial, for large-scale data this running time can be too slow. To overcome this issue, parallel algorithms for the Schulze method were proposed, and it turns out that some tasks of the method have efficient parallel solutions (e.g. checking if a candidate is a winner subject to the Schulze method is in NL [12]).

It is an interesting open question whether one can beat the known running time in the sequential setting, in particular without using fast matrix multiplication techniques. It is interesting how fast one can find all winners when the weighted majority graph is given, and whether the majority graph itself can be computed efficiently.

1.1 Our Contribution

Let m be the number of candidates, n the number of voters. Schulze [40] showed that the weighted majority graph can be constructed in $O(nm^2)$ time and that a set of all winners can be found in $O(m^3)$ time.

Observations. We first make several warm-up observations, connecting the Schulze method to problems in graph algorithms.

The first observation (Proposition 3.1) is that the problem of constructing the weighted majority graph can be reduced to the so called *Dominance Product* problem studied by Matoušek [32]. This gives a new running time for the weighted majority graph construction problem of $\tilde{O}(nm^{(1+\omega)/2} + n^{(2\omega-4)/(\omega-1)}m^2) \leq O(nm^{1.69} + n^{0.55}m^2)$, where $\omega < 2.373$ is the exponent of square matrix multiplication [1, 28, 48] and the $\tilde{O}$ notation hides subpolynomial factors. This running time always beats the previously published running time of $O(nm^2)$ when the number of voters and candidates is super-constant.

The second observation (Proposition 3.2) is folklore (see, e.g., the Wikipedia article [50]): computing the set of all winners (when the weighted majority graph is given) can easily be reduced to the so called All-Pairs Bottleneck Paths (APBP) problem: given a graph G, for every pair of vertices u, v, compute the maximum over all u-v paths of the minimum edge weight on the path (the so-called *bottleneck*). Using the fastest known algorithm for APBP [16], one can compute all winners in $\tilde{O}(m^{(3+\omega)/2}) \leq O(m^{2.69})$ time, easily beating the cubic running time. Actually, after solving APBP we obtain indirect comparisons between all pairs of candidates which can be used to construct a weak order over all candidates. This is useful in providing a fixed number of winners or as a preference aggregation method.

New improved algorithms. Although the above observations are of mostly theoretical interest, they do suggest that further algorithmic improvements can be possible. We turn our attention to obtaining even faster algorithms.

Our first main contribution is an almost *quadratic* time algorithm for finding all winners. This running time is substantially faster than all previously known algorithms. Furthermore, as the running time is almost linear in the size of the weighted majority graph, it is *essentially optimal*. The algorithm is combinatorial (it does not use heavy techniques that have large overheads like matrix multiplication), and is potentially of practical interest.

Theorem 1.1. *Given a weighted majority graph on m candidates, one can compute the set of all winners of the Schulze method in expected $O(m^2 \log^4(m))$ time.*

The theorem above appears as Theorem 5.3 in the body of the paper. As a warm-up to Theorem 5.3 we first present, in Theorem 4.1, a combinatorial algorithm for finding a single winner and then

generalize the approach to finding all winners. (Note that the problem of verifying that a particular candidate is a winner can easily be solved in $O(m^2)$ time using known algorithms, when given a weighted majority graph (see Proposition 3.3). However, computing the winners is a more difficult problem.)

Fine-grained lower bounds. While we were able to achieve an essentially optimal $\tilde{O}(m^2)$ time algorithm for finding all winners when given a weighted majority graph, computing the weighted majority graph itself seems more expensive. The fastest algorithm comes from our simple reduction to Dominance Product, resulting in a running time of $\tilde{O}(nm^{(1+\omega)/2} + n^{(2\omega-4)/(\omega-1)}m^2)$, or very slightly faster if $\omega > 2$ and one uses fast rectangular matrix multiplication.[5]

Typically, the number of voters n is no smaller than the number of candidates m. In this case, the running time for computing the majority graph using our reduction to Dominance Product is at least $\Omega(m^{2.5})$, regardless of the value of ω. Thus, if the input to the Schulze winner problem is not the weighted majority graph, but the preference profile, then the $\Omega(m^{2.5}) \gg m^2$ running time for computing the weighted majority graph is a significant bottleneck.

This leads to the following questions:

(1) Can one compute the weighted majority graph in $\tilde{O}(m^2)$ time, or at least in $O(m^{2.5-\varepsilon})$ time for some $\varepsilon > 0$?
(2) If not, can a winner be found in $O(m^{2.5-\varepsilon})$ time for some $\varepsilon > 0$, given the preference profile, potentially without computing the weighted majority graph explicitly?

We address the above questions through the lens of *fine-grained complexity* (see the surveys [37, 49]). The main goal of fine-grained complexity is to relate seemingly different problems via fine-grained reductions, hopefully proving equivalences. In recent years there has been an explosion of results in fine-grained complexity, showing that a large variety of problems are fine-grained reducible to each other. Sometimes fine-grained results can be viewed as hardness results, in the sense that a running time improvement for a problem can be considered unlikely. Fine-grained equivalences are especially valuable, as they often establish strong connections between seemingly unrelated problems.

For running time functions $a(n)$ and $b(n)$ for inputs of size (or measure[6]) n, we say that there is an (a, b)-fine-grained reduction from a problem A and to a problem B if for every $\varepsilon > 0$ there is a $\delta > 0$ and an algorithm that runs in $O(a(n)^{1-\delta})$ time and solves a size n instance of problem A using oracle calls to problem B of sizes $n_1, \ldots, n_k$ so that $\sum_{j=1}^{k} b(n_j)^{1-\varepsilon} \leq a(n)^{1-\delta}$. In other words, any improvement *in the exponent* over the $O(b(n))$ runtime for B on inputs of size n can be translated to an improvement in the exponent over the $O(a(n))$ runtime for A.

We relate questions (1) and (2) above to the complexity of the aforementioned Dominance Product problem and its relative *Dominating Pairs*. These problems have been studied within both algorithms (e.g. [32, 51]) and fine-grained complexity (e.g. [27, 30]).

In the Dominance Product problem we are given two $r \times r$ matrices A and B and we want to compute for every pair $i, j \in [r]$ the number of $k \in [r]$ for which $A[i, k] \leq B[k, j]$. In the Dominating Pairs problem we are also given two $r \times r$ matrices A and B, but we now want to decide whether there exists a pair $i, j \in [r]$ such that for all $k \in [r]$, $A[i, k] \leq B[k, j]$. In other words, we want to decide whether the Dominance Product of A and B contains an entry of value r.

[5] If $\omega = 2$, using rectangular matrix multiplication gives no improvement.
[6] In fact, in fine-grained complexity the running time is sometimes measured in terms of a different measure than the input size. For instance, the running time of graph problems is often measured in terms of the number of vertices instead of the true size of the graph which is in terms of the number of vertices and edges.

Both problems can be solved using the aforementioned $\tilde{O}(r^{(3+\omega)/2})$ time algorithm by Matoušek [32]. If $\omega > 2$, a slightly faster algorithm is possible using fast rectangular matrix multiplication [29], as shown by Yuster [51]. These running times are minimized at $\tilde{O}(r^{2.5})$ (in the case that $\omega = 2$).

We prove an equivalence between Dominance Product and the problem of computing the weighted majority graph, given a preference profile for the special case of n voters and $m = O(n)$ candidates. Note that even $m = n$ is a natural case—when a group of voters chooses a winner among themselves.

Theorem 1.2. *If there exists a $T(n)$ time algorithm for computing the weighted majority graph when there are n voters and $O(n)$ candidates, then there is an $O(T(r) + r^2 \log r)$ time algorithm for computing the Dominance Product of two $r \times r$ matrices. If there is a $T(r)$ time algorithm for computing the Dominance Product of two $r \times r$ matrices, then one can compute the weighted majority graph for n voters and $O(n)$ candidates in $O(T(n))$ time.*

First notice that since the matrices in the Dominance Product themselves have sizes $O(r^2)$, the $O(r^2 \log r)$ running time in the first part of the theorem is necessary, up to the $\log r$ factor. The first part of the theorem is proven in Theorem 6.1 and the second part follows from our simple observation described in Proposition 3.1.

As the $r^{2.5-o(1)}$ time bottleneck for solving Dominance Product and even Dominating Pairs has remained unchallenged for 30 years, it is believed that the two problems require $r^{2.5-o(1)}$ time (see, e.g., [30]). Due to Theorem 6.1 (in which $m = 2n$), we get that it would likely be very challenging to obtain an $O(m^{2.5-\varepsilon})$ time algorithm for $\varepsilon > 0$ for computing the weighted majority graph, even if $\omega = 2$.

We have shown that constructing the weighted majority graph is likely an expensive task. It could still be, however, that one can find a winner without explicitly constructing the weighted majority graph, as per our question (2) above. Our final result relates question (2) to the Dominating Pairs problem:

Theorem 1.3. *Suppose that there is an $O(T(n))$ time algorithm that, given n voters with preferences over $O(n)$ candidates, can test whether a given candidate is a winner of the Schulze method. Then there is an $O(T(r) + r^2 \log r)$ time algorithm for the Dominating Pairs problem for two $r \times r$ matrices.*

The theorem above appears as Theorem 6.2 in the body of the paper. Because in the proof we have $m = \Theta(n)$, our result shows that even testing whether a candidate is a winner in $O(m^{2.5-\varepsilon})$ time for $\varepsilon > 0$ would be a challenge under the plausible hypothesis that Dominating Pairs has no improved algorithms even in the case that $\omega = 2$. Under this hypothesis, we might as well compute the weighted majority graph and find all winners using Theorem 5.3, as it would take roughly the same time.

Our paper is the *first* to bring fine-grained complexity into the field of computational social choice (see [2, 9, 19] for a description of computational social choice topics). Related areas such as Fixed Parameter Tractability (FPT)[7] and hardness of approximation[8] have already gained significant traction in computational social choice. See, e.g., the following surveys on FPT results and further challenges in computational social choice [10, 14, 22], and, e.g., the following papers on hardness of approximation in bribery problems [21], multiwinner elections [3, 4, 17, 43] and a survey on fair-division with indivisible goods [31].

[7] Where some standard assumptions are: FPT ≠ W[1] and the Exponential Time Hypothesis. For more about parameterized complexity see, e.g., [13, 15].

[8] Here some standard assumptions are P ≠ NP , the Unique Games Conjecture [26], and a recent Gap-Exponential Time Hypothesis [23].

While these related areas do discuss running time, they do not focus on the exact running time complexity, and in particular, say nothing about lower bounds for problems that already have fixed polynomial time algorithms. By bringing fine-grained complexity into the picture we can identify which voting protocols have potentially practical algorithms, and those whose complexity is, say at least cubic, and hence are likely impractical for large numbers of candidates and/or voters.

Our results provide reductions and equivalences that are even tighter than typical fine-grained reductions, in that the running times are preserved up to small additive terms and constant multiplicative factors.

1.2 Related Work

Below we present a few selected computational topics considered in the context of the Schulze method. For a detailed survey of related work and historical notes, we refer the reader to a technical report by Schulze [41]. In particular, we will not discuss the axiomatic properties of the Schulze method; a comprehensive study of this appears in [40, Section 4] and its extended version [41, Section 4].

Parallel computing. Csar et al. [12] considered large-scale data sets and claimed that a straightforward implementation of the Schulze method does not scale well for large elections. Because of this, they developed an algorithm in the massively parallel computation model. They showed that the Schulze winner determination problem is NL-complete. The containment in NL implies that the method is well-suited for parallel computation. Csar et al. [12] also conducted experiments to show that their algorithm scales well with additional computational resources.

Strategic behavior. There are 3 basic types of strategic behaviors: manipulation, control and bribery.

Manipulation is defined as the casting of non-truthful preference ballots by a voter or a coalition of voters in order to change the outcome of an election (to make a preferred candidate a winner is called "constructive manipulation", and to make a particular candidate a loser is called "destructive manipulation").

Control has eight basic variants, each defined by picking a choice from each of the following three pairs of groups: {constructive, destructive}, {adding, deleting}, {votes, alternatives}.; e.g. "constructive control by deleting votes".

Bribery allows one to completely change votes, however, there is a cost of changing each vote, so that the number of affected votes is to be minimized. There is a constructive and a destructive version of bribery.

Parkes and Xia [35] showed that the Schulze method is vulnerable (polynomial-time solvable) to constructive manipulation by a single voter and destructive manipulation by a coalition. From the other side, the Schulze method is resistant (NP-hard to compute a solution) to, e.g., constructive control by adding alternatives (i.e., it is NP-hard to find a subset of additional alternatives such that a preferred candidate becomes a winner); control by adding/deleting votes; and both cases of bribery. For more specific results on computational issues of strategic behaviors under the Schulze method see [24, 34]. Furthermore, FPT algorithms for NP-hard types of strategic behaviors were studied by Hemaspaandra et al. [25].

Margin of victory. The Schulze method was studied in terms of the *margin of victory*, which is the minimum number of modified votes resulting in a change to the set of winners. This is a similar concept to bribery, but here we only care about the stability of a solution, not about making a candidate a winner/loser. This concept is used to measure the robustness of voting systems due to errors or frauds. The higher the margin of victory is, the more robust the election is. Reisch

et al. [36] showed, e.g., that computing the margin of victory for the Schulze method is NP-hard (using the NP-hardness proof for destructive bribery due to Parkes and Xia [35]), so that evaluating the robustness of the election might be difficult.

2 PRELIMINARIES

To define the method formally we need to introduce some notations first.

A weak order (or *total preorder*) $\succeq$ is a binary relation on a set S that is transitive and connex (complete), i.e., (1) if $x \succeq y$ and $y \succeq z$ then $x \succeq z$; (2) for all $x, y \in S$ we have $x \succeq y$ or $y \succeq x$. We define a strict part $\succ$ of a weak order $\succeq$ by: $x \succ y$ iff $x \succeq y$ and $y \not\succeq x$. Then $\succ$ is a *strict weak order*, and the Schulze method was originally defined using it [40].

We denote the set of voters by N, and the set of alternatives (candidates) by A, where $|N| = n$ and $|A| = m$. *A preference profile* P (or a multiset of votes) is a list $(\succeq_a)_{a \in N}$ of weak orders over a set of alternatives A. $u \succ_a v$ ($u \succeq_a v$) means that a voter $a \in N$ strictly (weakly respectively) prefers alternative $u \in A$ to alternative $v \in A \setminus \{u\}$.

We define $M(u, v)$, where $u, v \in A$, as the number of voters that strictly prefer u over v, i.e., $M(u, v) = |\{a \in N : u \succ_a v\}|$.

For a given preference profile P, the *weighted majority graph* G is defined as follows: the vertices of G is the set of alternatives A, and for every $u, v \in A$ we have directed edges (u, v) and (v, u) with associated weights $w(u, v) = M(u, v) - M(v, u)$ and $w(v, u) = M(v, u) - M(u, v)$. We also call $w(u, v)$ *the strength of the link* (u, v). We note that Schulze [40] defined the strength of the link in a more general way, but he proposed to use $w(u, v)$ defined above, as it is the most intuitive notion of strength. Indeed, this is the most popular strength of the link definition used in the literature [12, 24, 25, 34–36].

We define the *strength of indirect comparison* of alternative $u \in A$ versus alternative $v \in A \setminus \{u\}$, denoted by $B_G(u, v) \in \{0, 1, \ldots, n\}$, as the weight of *the maximum bottleneck path* (also called widest path) from u to v. $B_G(u, v)$ is the maximum *width* or *bottleneck* of any path from u to v, where the width/bottleneck of a path is equal to its minimum edge weight. Formally $B_G(u, v) = \max_{(x_1=u, x_2, \ldots, x_{k-1}, x_k=v): x_j \in A} \min_{i \in \{2,3,\ldots,k\}} \{w(x_{i-1}, x_i)\}$. (The maximum is well-defined because widest paths are simple without loss of generality, similar to shortest paths.) If G is clear from context, we sometimes use $B(u, v)$ as well.

A set of winners $\mathcal{W}$ in the Schulze method consists of all $u \in A$ such that for every $v \in A$ we have $B_G(u, v) \geq B_G(v, u)$. It is known that there always exists a winner, i.e., $\mathcal{W} \neq \emptyset$ [40]. We call the elements of $\mathcal{W}$ *the Schulze winners*. By $\mathcal{W}(G)$ we denote a set of winners for a given weighted majority graph G.

Schulze-AllWinners is the problem of finding all winners (i.e., the set $\mathcal{W}$), and Schulze-Winner is that of finding a winner (i.e., an element from $\mathcal{W}$). The decision version of the problem, Schulze-WinnerDetermination [12], asks whether a given candidate is a winner.

The Schulze method was designed as a single winner election rule, but using it we can construct a weak order over the set of all alternatives. Hence the Schulze method may be seen as a preference aggregation method. For this we define the relation R such that $(u, v) \in R$ if and only if $B_G(u, v) > B_G(v, u)$. Note that the Schulze winners are top-ranked alternatives in the order derived from R. Also note that a proof of $\mathcal{W} \neq \emptyset$ follows from transitivity of R [40, Section 4.1].

We use ω to denote the smallest real number such that one can multiply two $r \times r$ matrices in $O(r^{\omega+\epsilon})$ time for every $\epsilon > 0$. Currently we know that $2 \leq \omega < 2.373$ [1, 28, 48]. We also use $\mathcal{M}(a, b, c)$ to denote the fastest running time for multiplying an $a \times b$ matrix and a $b \times c$ matrix.

For a graph G and a subset of vertices $U \subseteq V(G)$, we use $G[U]$ to denote the subgraph induced by the vertex set U. We use *strongly-connected-component (SCC) of a vertex* v to denote the set of vertices that can both reach and be reached from v.

3 WARM-UP

Proposition 3.1. *For a preference profile with m candidates and n voters, we can compute the weighted majority graph in $\min_s \tilde{O}(\mathcal{M}(m, sn, m) + nm^2/s)$ time.*

Proof. We will compute the weighted majority graph using the algorithm for Dominance Product [51], which runs in time $\min_s \tilde{O}(\mathcal{M}(m, sn, m) + nm^2/s)$.

Given m candidates and n voters, we will create the matrices A and B as follows. Since the preference list of each voter a is a weak order, we can associate an integer value $f_{a,u}$ with each voter a and each candidate u such that $u \succ_a v$ if and only if $f_{a,u} < f_{a,v}$ for any two candidates u, v (these values can be the ranks in the sorted order of a's preference list). We then set $A_{u,a} = f_{a,u}$ for any pair consisting of a voter a and a candidate u; we set $B_{a,v} = f_{a,v} - \frac{1}{2}$ for any pair consisting of a voter a and a candidate v. Then the Dominance Product C between A and B is such that

$$C_{u,v} = \left|\left\{a \in N : f_{a,u} \leq f_{a,v} - \frac{1}{2}\right\}\right|,$$

which equals exactly $M(u, v)$ by the definition of the integer values f. Using M, we can compute the weighted majority graph in $O(m^2)$ time.

Therefore, the bottleneck of the algorithm is the Dominance Product problem, which has running time $\min_s \tilde{O}(\mathcal{M}(m, sn, m) + nm^2/s)$. □

Suppose that $n \geq m^{(\omega-1)/2}$. Then we set s to a value such that $sn \geq m$. In this case we can compute the product between an $m \times (sn)$ matrix and an $(sn) \times m$ matrix by first splitting the second dimension of the first matrix to roughly $\frac{sn}{m}$ pieces, and then using fast square matrix multiplication to compute the product between each pair of corresponding pieces. Thus, we can upper bound $\mathcal{M}(m, sn, m)$ by $\tilde{O}((sn/m) \cdot m^\omega)$. By setting $s = m^{(3-\omega)/2}$, the running time of Proposition 3.1 becomes $\tilde{O}(nm^{(1+\omega)/2}) \leq O(nm^{1.69})$.

If $n \leq m^{(\omega-1)/2}$, then we set s so that $sn \leq m$. We can similarly bound $\mathcal{M}(m, sn, m)$ by $\tilde{O}((m/sn)^2 \cdot (sn)^\omega)$. By setting $s = n^{(3-\omega)/(\omega-1)}$, the running time of Proposition 3.1 becomes $\tilde{O}(n^{(2\omega-4)/(\omega-1)} \cdot m^2) \leq O(n^{0.55}m^2)$.

Overall, the running time of Proposition 3.1 is always upper bounded by $O(nm^{1.69} + n^{0.55}m^2)$. If $\omega > 2$ the running time can be improved slightly by using the best known bounds on rectangular matrix multiplication [29] to compute $\mathcal{M}(m, sn, m)$ in both cases. We will not go into detail here because the setting of s here would depend on how n and m are related, and the current best bounds on rectangular matrix multiplication are obtained by using numerical solvers for each setting of the matrix dimensions. As mentioned in the Our Contribution subsection, if $n \geq m$, the running time is always greater than $m^{2.5-o(1)}$, regardless of the value of ω and the use of rectangular matrix multiplication.

The next two propositions are folklore [50]. We include their proofs here for completeness.

Proposition 3.2. *Given a weighted majority graph on m candidates,* Schulze-AllWinners *can be solved in $\tilde{O}(m^{(3+\omega)/2})$ time.*

Proof. Given a weighted majority graph, all values $B_G(u, v)$ can be obtained via a single All-Pairs Bottleneck Paths (APBP) computation. APBP has been well-studied by the graph algorithms community [16, 42, 47] and its current best running time is $\tilde{O}(m^{\frac{3+\omega}{2}})$ for an m-vertex graph [16]. After we compute $B_G(u, v)$, it only takes $O(m^2)$ additional time to determine all Schulze winners. □

Proposition 3.3. *Given a weighted majority graph on m candidates and a particular candidate v, we can solve* Schulze-WinnerDetermination *in $O(m^2)$ time.*

Proof. Since we only need to verify if some candidate v is a winner, it suffices to compute $B_G(v,u)$ and $B_G(u,v)$ for all $u \in V$. Computing $B_G(v,u)$ is exactly the problem of computing Single-Source Bottleneck Paths (SSBP). In a dense graph with $\Theta(m^2)$ edges, the best algorithm for SSBP runs in $O(m^2)$ time by using Dijkstra's algorithm augmented with Fibonacci heap. To compute $B_G(u,v)$, we can reverse the directions of all edges in the graph and compute another SSBP. Using $B_G(v,u)$ and $B_G(u,v)$ for all $u \in V$, it only takes $O(m)$ time to determine if v is a winner. □

4 FINDING A WINNER

In this section we show that given a weighted majority graph, we can find a Schulze winner in almost quadratic time.

Theorem 4.1. *Given a weighted majority graph on m candidates,* Schulze-Winner *can be solved in expected $O(m^2 \log^4(m))$ time.*

We will use the following decremental SCC algorithm of Bernstein et al. [6].

Theorem 4.2. *(Bernstein et al. [6]) Given a graph $G = (V, E)$, we can maintain a data structure that supports the following operations:*

- `DELETE-EDGE`(u,v)*: Deletes the edge (u,v) from the graph.*
- `SAME-SCC`(u,v)*: Returns whether u and v are in the same SCC.*

The data structure runs in total expected $O(|E| \log^4 |V|)$ time for all deletions and worst-case $O(1)$ time for each query. The bound holds against an oblivious adaptive adversary.

Since the answer for each `SAME-SCC` query is unique, an oblivious adaptive adversary is equivalent to a non-adaptive adversary in this setting.

Our algorithm is simple and can be described in ten lines as shown in Algorithm 1.

ALGORITHM 1: Schulze-Winner $(G = (V, E))$

```
Data: G = (V, E)
Result: One winner of graph G.
1  Sort E by weights in increasing order;
2  x ← an arbitrary vertex in V;
3  for (u, v) ∈ E do
4  |   flag ← SAME-SCC(u, x) ∧ SAME-SCC(v, x);
5  |   DELETE-EDGE(u, v);
6  |   if SAME-SCC(u, v) = False and flag then
7  |   |   x ← v;
8  |   end
9  end
10 return x;
```

Lemma 4.3. *Any time Algorithm 1 finishes Line 2 or Line 8, any winner of the subgraph induced by the SCC of x is a winner of the original graph G.*

Proof. Even though Algorithm 1 seems to operate directly on graph G, in the proof we assume it operates on a copy of the graph and thus the original graph still has all its edges. Thus, we always use G to denote the original graph with no edges removed in this proof.

When the algorithm finishes Line 2, all edges in the graph still exist. Since the graph is complete, the SCC of x is exactly V. Therefore, any winner of the subgraph induced by the SCC of x is a winner of the whole graph.

In the remainder of this proof, we will use U to denote the set of vertices in the same SCC as x in the graph where we remove all the edges in sorted list E before (u, v) (i.e., before we execute Line 5). We also use U' to denote the set of vertices in the same SCC as x in the graph where we remove all the edges up to (u, v) and we considered updating x in Line 7 under certain conditions (i.e., after we execute Line 8).

We prove the second part of the lemma by induction. Suppose $\mathcal{W}(G[U]) \subseteq \mathcal{W}(G)$, and we need to prove $\mathcal{W}(G[U']) \subseteq \mathcal{W}(G)$. To achieve this, it suffices to show $\mathcal{W}(G[U']) \subseteq \mathcal{W}(G[U])$.

First of all, if *flag* is set to *False*, then either u or v is not in U, so the edge (u, v) does not lie entirely in the set U. Thus, deleting the edge (u, v) does not change the SCC of x. Also, since *flag* is set to *False*, the algorithm will not execute Line 7, so the value of x also stays the same. Thus, since both x and the SCC of x stays unchanged, $U = U'$ and clearly $\mathcal{W}(G[U']) \subseteq \mathcal{W}(G[U])$.

Secondly, suppose the SAME-SCC(u, v) check at Line 6 is *True*. This means u and v are still in the same SCC after deleting (u, v), so in particular, u can still reach v. Thus, even though we just deleted the edge (u, v), the connectivity of the graph is unaffected. Thus, the SCC of x is unchanged. Also, the SAME-SCC(u, v) check is *True*, so we will not execute Line 7 in this iteration. Therefore, similar to the previous case, $\mathcal{W}(G[U']) \subseteq \mathcal{W}(G[U])$.

The only case remaining is when *flag* is set to *True* and the SAME-SCC(u, v) check at Line 6 is *False*. Let y be any winner of the graph $G[U']$. We will show that y is a winner of $G[U]$ as well, and by the induction hypothesis $\mathcal{W}(G[U]) \subseteq \mathcal{W}(G)$, y will be a winner of G.

We have to show that $B_{G[U]}(y, z) \geq B_{G[U]}(z, y)$ for every $z \in U$. There are two cases depending on where z is included.

(1) $z \in U'$. Since y is a winner of $G[U']$, we have $B_{G[U']}(y, z) \geq B_{G[U']}(z, y)$. Now consider any path from y to z in $G[U]$. If this path ever touches any vertex z' outside of U', then it must use an edge that is already deleted, since otherwise, $y \in U'$ can reach z' and z' can reach $z \in U'$ so z' must also be in the SCC U, which leads to a contradiction. Thus, this path must use an edge of weight at most $w(u, v)$, so this path has a bottleneck at most $w(u, v)$.[9] However, since U' is strongly connected, and all edges that are still present have weights at least $w(u, v)$, so $B_{G[U']}(y, z) \geq w(u, v)$. Thus, $B_{G[U]}(y, z) = B_{G[U']}(y, z)$ since if a path leaves U' then its bottleneck is at most $w(u, v) \leq B_{G[U']}(y, z)$. Similarly, $B_{G[U]}(z, y) = B_{G[U']}(z, y)$. Therefore, $B_{G[U]}(y, z) \geq B_{G[U]}(z, y)$.

(2) $z \in U \setminus U'$. We first show that y can reach z using only edges that are not yet deleted right after we delete (u, v). First, since $y \in U'$, y can reach v. Since U is an SCC before we delete (u, v), v can reach all vertices in U before we delete (u, v). However, any simple path from v to some other vertex in U does not use the edge (u, v), so v can still reach all other vertices in U even after deleting (u, v). Therefore, y can reach z through v, and thus $B_{G[U]}(y, z) \geq w(u, v)$. On the other hand, z cannot reach y using not yet deleted edges since otherwise z will be in the same SCC as y. Therefore, $B_{G[U]}(z, y) \leq w(u, v) \leq B_{G[U]}(y, z)$.

□

Lemma 4.4. *Algorithm 1 always returns a winner of G.*

Proof. By Lemma 4.3, at Line 10 of Algorithm 1, any winner of the graph induced by the SCC of x is a winner of G. Furthermore, at Line 10, we already deleted all edges in the graph, so the SCC of x just contains x itself. Thus, x is a winner of the graph induced by the SCC of x and, by Lemma 4.3, x is a winner of G. □

Lemma 4.5. *Algorithm 1 runs in expected $O(m^2 \log^4(m))$ time.*

[9] Recall that $G[U]$ is an induced subgraph of G in which all the edges are present—also these removed before removing (u, v)—hence a bottleneck of the considered path can be strictly smaller than $w(u, v)$.

Proof. Sorting the edge list E takes $O(m^2 \log m)$ time. Calling DELETE-EDGE(u, v) for all $(u, v) \in E$ takes expected $O(m^2 \log^4(m))$ time by Theorem 4.2 since we fix the order to delete the edges in advance and thus our algorithm behaves like a non-adaptive adversary. Also, each call of SAME-SCC takes $O(1)$ time by Theorem 4.2. All remaining components of Algorithm 1 takes $O(m^2)$ time. Therefore, Algorithm 1 runs in expected $O(m^2 \log^4(m))$ time. □

Proof of Theorem 4.1. The theorem follows immediately from Lemma 4.4 and Lemma 4.5. □

5 FINDING ALL WINNERS

In order to compute all winners, we need to augment the decremental SCC algorithm with more information. This could be done in a black-box way.

In this section, we define the in-degree of an SCC U as $|\{(v, u) \in E : v \notin U, u \in U\}|$.

Corollary 5.1. *Given a graph $G = (V, E)$, we can maintain a data structure that keeps the following:*

- *A set $\mathcal{S}$ containing all IDs of SCCs of the graph.*
- *A map $\mathcal{D}$ from the IDs of SCCs of the graph to the in-degrees of the SCCs.*
- *A map $\mathcal{SCC}$ from vertices of the graph to the ID of the SCCs they are in.*

The data structure also supports the following operations:

- DELETE-EDGE(u, v)*: Deletes the edge (u, v) from the graph. Additionally, the data structure needs to return a list of new SCCs being created.*
- SAME-SCC(u, v)*: Returns whether u and v are in the same SCC.*

The data structure runs in total expected $O(|E| \log^4 |V|)$ time for all deletions and worst-case $O(1)$ query time. The bound holds against a non-adaptive adversary.

Proof. The high-level strategy of the algorithm is to first use the data structure from [6], and then use the "removing small from large" strategy used in, e.g., [6, 20, 45] to explicitly maintain all the SCCs.

Initially, it is easy to set up the set $\mathcal{S}$ and the maps $\mathcal{D}$ and $\mathcal{SCC}$. We also initialize the data structure of Bernstein et al. [6]. For each vertex v, we create a list of vertices $\mathcal{N}(v)$ that contain all the neighbors of v considering edges in both directions.

For each DELETE-EDGE(u, v) operation, if u and v are not in the same SCC, it suffices to update the map $\mathcal{D}$. If u and v are in the same SCC both before and after deleting the edge, we do not need to update $\mathcal{S}$, $\mathcal{D}$ or $\mathcal{SCC}$.

The trickiest case is when u and v are in the same SCC U before deleting (u, v), but not in the same SCC after deleting (u, v). In this case, we use the "removing small from large" method to find all new SCCs. We will explicitly enumerate all the vertices in every new SCC except one SCC with the largest total degree of the vertices in it.

We first delete edge (u, v) in the data structure of Bernstein et al. [6]. Then we create a list of vertices L, which initially only contains u and v. Each time, we repeatedly take a vertex out of L until we find one vertex x whose new SCC has not been found. Then we repeatedly take another vertex out of L until we find one vertex y which is not in the same new SCC as x and whose new SCC has not been found. If we could not find such a vertex y before L becomes empty, then we end the process. We interleave two breadth-first-searches (BFSes) from vertex x and vertex y by using neighbors stored in $\mathcal{N}$, but exploring only those vertices which are in the same new SCC as x or y respectively by calling SAME-SCC. Note that these BFSes use all edges in the original graph and ignore the edge directions. As soon as one of the BFSes finishes (the SCC with the smaller total degree will finish sooner), we stop the other BFS as well. Without loss of generality, we assume

the BFS from x finishes sooner. In this case we have a list of all vertices in the same SCC with x. For every vertex w in this list, we enumerate all vertices z in $\mathcal{N}(w)$, and add z to L if z belonged to the SCC U (we could call the map $\mathcal{SCC}$ for checking this, since it is not updated after (u, v) gets deleted). Finally we put y back to L and repeat the above process.

We show that the process above will find all new SCCs except the one SCC U' with the largest total degree. To show this, it suffices to show that all vertices in the induced subgraph $G[U \setminus U']$ are either weakly connected to v or weakly connected to u via a path inside $G[U \setminus U']$. Let z be any vertex in $U \setminus U'$. Since before we delete (u, v), U is an SCC, so there is a simple path p_1 inside U from v to z that only uses edges still in the graph including (u, v). However, since p_1 starts from v, and it is simple, it will not use (u, v), so p_1 still exists even after deleting (u, v). Similarly, there is a path from z to u that still exists after deleting (u, v). If one of p_1 or p_2 does not enter U', then we are done. Otherwise, both p_1 and p_2 touch U', so z can both reach and be reached from U'. Hence, z also belongs to the SCC U', a contradiction. Therefore, our algorithm will find all but one new SCCs.

Since we can list all vertices in all but one new SCCs, it is then easy to update $\mathcal{S}, \mathcal{D}$ and $\mathcal{SCC}$. However, we should note that the SCC U' with the largest total degrees will have the same ID as the old SCC U, since we cannot afford to update the map $\mathcal{SCC}$ for every vertex in U'. It is also easy to return a list of new SCCs being created.

Now we analyze the running time of the BFSes, which is quite standard. Each time we find an SCC with total degree D, we will pay $O(D)$ in the BFS and $O(D \log |V|)$ for updating $\mathcal{S}, \mathcal{D}$ and $\mathcal{SCC}$. We also know that we are taking it from an old SCC of total degree at least $2D$. Therefore, each edge that contributes to D can be taken out at most $O(\log |V|)$ times, since each time the SCC that contains one endpoint of the edge must halve in total degree. Therefore, the total cost of BFSes is $O(|E| \log |V|)$, and there is another $O(\log |V|)$ factor for updating the necessary data structures. Thus, it is clear that the running time is dominated by the $O(|E| \log^4 |V|)$ factor from [6]. □

Using Corollary 5.1, the algorithm for finding all winners is described in Algorithm 2.

We also describe the algorithm in text for more intuition. The algorithm maintains a set C that contains all candidate SCCs that could contain winners. In increasing order of weight w, the algorithm deletes all edges of weight w in a batch. If a candidate SCC splits into multiple smaller SCCs, we remove this candidate SCC from C and add those small SCCs whose in-degrees are 0s back to C. Eventually, all SCCs contain single vertices, and we return the single vertices in those candidate SCCs. Now we prove the correctness of the algorithm via the following lemma.

Lemma 5.2. *Before and after each iteration in the "for" loop in Line* 3 *of Algorithm 2, we have*

$$\mathcal{W}(G) = \bigcup_{scc \in C} \mathcal{W}(G[scc]),$$

where G denotes the original graph with no edge removed.

Proof. Before running any iteration of the "for" loop, C only contains the whole graph, so the equality is clearly true.

Now we prove the equality by induction. Suppose that the equality is true before we run the "for" loop for value w. For each vertex set $U \in C$ that was an SCC before deleting all edges of weight w, we will split it into multiple SCCs after deleting those edges, and put those $U'_1, \ldots, U'_t$ whose in-degrees are 0s back to C. Thus, it suffices to show that

$$\mathcal{W}(G[U]) = \bigcup_{i=1}^{t} \mathcal{W}(G[U'_i]).$$

ALGORITHM 2: Schulze-AllWinners ($G = (V, E)$)

Data: $G = (V, E)$
Result: All winners in graph G.

```
Let W be a sorted list of all distinct weights of E;
Let C be a set containing a single SCC, the whole graph;
for w ∈ W do
    Let L be an empty list;
    for every edge (u, v) of weight w do
        scc ← SCC(u);
        flag ← SAME-SCC(u, v);
        l ← DELETE-EDGE(u, v);
        if SAME-SCC(u, v) = False and flag and scc ∈ C then
            Remove scc from C;
            Add every SCC in l to C;
            Add every SCC in l to L;
        end
    end
    for scc ∈ L do
        if D(scc) > 0 then
            Remove scc from C;
        end
    end
end
return {v ∈ V : SCC(v) ∈ C};
```

Let $x \in U$ be an arbitrary winner of $G[U]$. First, suppose the in-degree of $\mathcal{SCC}(x)$ after deleting the weight w edges are nonzero. In this case, there exists another vertex $y \in U \setminus \mathcal{SCC}(x)$ that can reach x. Therefore, $B_{G[U]}(y, x) > w$. However, x cannot reach y since otherwise y is in the same SCC as x, so $B_{G[U]}(x, y) \leq w$, and thus x cannot be a winner of $G[U]$, a contradiction. Therefore, we can assume $x \in U_i'$ for some $1 \leq i \leq t$. For any $y \in U_i'$, consider the widest path between x and y. Since x and y belong to the same SCC after removing weight w edges, $B_{G[U]}(x, y) > w$ and $B_{G[U]}(y, x) > w$. Also, if any path leaves U_i' and comes back, the bottleneck of that path is upper bounded by w, so an optimal path will not leave U_i'. Therefore, $B_{G[U_i']}(x, y) = B_{G[U]}(x, y)$ and $B_{G[U_i']}(y, x) = B_{G[U]}(y, x)$. Since x is a winner of $G[U]$, $B_{G[U]}(x, y) \geq B_{G[U]}(y, x)$. Therefore, $B_{G[U_i']}(x, y) \geq B_{G[U_i']}(y, x)$. We conclude that x is a winner of $G[U_i']$. Hence, $\mathcal{W}(G[U]) \subseteq \bigcup_{i=1}^{t} \mathcal{W}(G[U_i'])$.

Now we show the other direction. For any $1 \leq i \leq t$, let x be an arbitrary winner of $G[U_i']$. For any $y \in U$, we want to show that $B_{G[U]}(x, y) \geq B_{G[U]}(y, x)$. First, if $y \notin U_i'$, then there is no path from y to x using only edges of weight greater than w, because the in-degree of U_i' is 0. Therefore, $B_{G[U]}(y, x) \leq w$. On the other hand, since $x, y \in U$ and U is strongly connected using edges of weight up to w, we have $B_{G[U]}(x, y) \geq w$. Thus, $B_{G[U]}(x, y) \geq B_{G[U]}(y, x)$. Secondly, if $y \in U_i'$, then the widest paths from x to y and from y to x are completely inside U_i' by a previous argument, so $B_{G[U_i']}(x, y) = B_{G[U]}(x, y)$ and $B_{G[U_i']}(y, x) = B_{G[U]}(y, x)$. Since x is a winner of $G[U_i']$, $B_{G[U_i']}(x, y) \geq B_{G[U_i']}(y, x)$. Therefore, $B_{G[U]}(x, y) \geq B_{G[U]}(y, x)$. We conclude that x is a winner of $G[U]$ and hence $\mathcal{W}(G[U]) \supseteq \bigcup_{i=1}^{t} \mathcal{W}(G[U_i'])$.

In conclusion, we showed that $\mathcal{W}(G[U]) = \bigcup_{i=1}^{t} \mathcal{W}(G[U_i'])$, which is sufficient for the induction to complete. □

Theorem 5.3. *Given a weighted majority graph on m candidates,* Schulze-AllWinners *can be solved in expected* $O(m^2 \log^4(m))$ *time.*

Proof. The algorithm is shown in Algorithm 2. By Lemma 5.2, at Line 21, we have $\mathcal{W}(G) = \bigcup_{scc \in C} \mathcal{W}(G[scc])$. Furthermore, since all edges are deleted in the graph and thus all SCCs contain single vertices at Line 21, $\mathcal{W}(G[scc]) = V(scc)$. Therefore, $\mathcal{W}(G) = \{v \in V : \mathcal{SCC}(v) \in C\}$ and thus the returned result is correct.

Algorithm 2 clearly runs in expected $O(m^2 \log^4(m))$ time. □

6 LOWER BOUNDS

Theorem 6.1. *If there exists a* $T(n)$ *time algorithm for computing all the edge weights of the weighted majority graph when there are n voters and* $2n$ *candidates, then there is an* $O(T(r) + r^2 \log r)$ *time algorithm for computing the Dominance Product of two* $r \times r$ *matrices.*

Proof. Given two $r \times r$ matrices A and B, we will compute their Dominance Product C, where $C_{i,j} = \left|k \in [r] : A_{i,k} \leq B_{k,j}\right|$, using the assumed $T(n)$ time algorithm for computing the weighted majority graph.

We first pre-process the two matrices so that all entries are distinct. We could achieve this by first putting all the entries to a list, and then sorting the list. If there is a tie between several elements, we always sort an element corresponding to an entry of A earlier than an element corresponding to an entry of B. Then we can replace all entries with their position in the sorted list. The Dominance Product of A and B clearly does not change. This pre-processing only takes $O(r^2 \log r)$ time.

In our construction, there are $m = 2r$ candidates labeled as $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$. We will create one voter for each $k \in [r]$, so there are a total of $n = r$ voters. The k-th voter associates a number $A_{i,k}$ with candidate u_i for every $i \in [r]$ and a number $B_{k,j}$ with candidate v_j for every $j \in [r]$. Then the k-th voter prefers a candidate x over a candidate y if and only if the associated number of candidate x is smaller than that of candidate y. Since all entries of these two matrices are distinct, the preference order of each voter is linear (i.e., for every voter i and any distinct candidates x and y we have either $x \succ_i y$ or $x \prec_i y$).

We show that the value $M(u_i, v_j)$ corresponding to preference profile above equals $C_{i,j}$. In fact, if the k-th voter prefers u_i to v_j, then its associated number with u_i is smaller than its associated number with v_j, i.e., $A_{i,k} < B_{k,j}$. Therefore, $M(u_i, v_j)$ equals the number of k such that $A_{i,k} < B_{k,j}$. Since all entries of the two matrices are distinct, $A_{i,k} < B_{k,j}$ if and only if $A_{i,k} \leq B_{k,j}$. Thus, $M(u_i, v_j) = C_{i,j}$.

In the weighted majority graph built on linear orders only, the edge weight $w(u_i, v_j)$ equals to $M(u_i, v_j) - M(v_j, u_i) = 2M(u_i, v_j) - r = 2C_{i,j} - r$, so we could compute $C_{i,j}$ from $w(u_i, v_j)$ via $C_{i,j} = (w(u_i, v_j) + r)/2$. Therefore, we can call the $T(n) = T(r)$ time algorithm for computing the weighted majority graph and get the Dominance Product C from the edge weights of the graph easily. □

Next, we will show the conditional hardness for Schulze-WinnerDetermination by reducing from the Dominating Pairs problem. Note that in the Dominating Pairs problem, we essentially want to test if the Dominance Product of two $r \times r$ matrices contains an entry of value r.

Theorem 6.2. *Suppose that there is an* $O(T(n))$ *time algorithm that, given n voters with preferences over* $\Theta(n)$ *candidates, can solve* Schulze-WinnerDetermination. *Then there is an* $O(T(r) + r^2 \log r)$ *time algorithm for the Dominating Pairs problem for two* $r \times r$ *matrices.*

Proof. Given two $r \times r$ matrices A and B, we will create a preference profile on $m = 2r + 2$ candidates $u_1, \ldots, u_r, v_1, \ldots, v_r, W, W'$ and $n = 10r - 2$ voters. For technical reasons, we assume

all entries in the Dominance Product C between A and B are positive. This can easily be addressed by padding an additional column with 0s to matrix A and a corresponding row with 1s to matrix B. We can perform additional padding to make A and B square again: add a row to A that contains entries strictly greater than any entry in B (except the last one entry in this row which equals to 0); add a column to B that contains entries strictly smaller than any entry in A (except the last one entry in this column which equals to 1). With an $O(r^2 \log r)$ time pre-processing, we can assume all the entries of the matrices A and B are distinct (similarly as in the proof of Theorem 6.1).

The first r voters will look like the voters in the proof of Theorem 6.1. Specifically, the k-th voter associates a number $A_{i,k}$ with candidate u_i for every $i \in [r]$, and a number $B_{k,j}$ with candidate v_j for every $j \in [r]$. Then the k-th voter prefers a candidate x over a candidate y if and only if the associated number of candidate x is smaller than that of candidate y. Moreover, the first r voters always prefer W the least and prefer W' the second least.

The next r voters will be similar to the first r voters. For any $k \in [r]$, the preference list of the $(k+r)$-th voter is the same as the preference list of the k-th voter, except that $(k+r)$-th voter always prefers W the most and prefer W' the second most.

Let us consider how the first $2r$ voters will affect the values of M. From the first $2r$ voters, we add $2C_{i,j}$ to $M(u_i, v_j)$ for all $i, j \in [r]$, add $2r - 2C_{i,j}$ to $M(v_j, u_i)$ for all $i, j \in [r]$ and add r to all edges with one endpoint being W or W'. We call these edges *important* as other edge weights will not be important in our analysis.

The preference lists between the $(2r+1)$-th voter and the $(3r-1)$-th voter are all the following:

$$u_1 \prec u_2 \prec \cdots \prec u_r \prec W \prec v_1 \prec \cdots \prec v_r \prec W'.$$

The preference lists between the $(3r)$-th voter and the $(4r-2)$-th voter are all the following:

$$W' \prec v_r \prec \cdots \prec v_1 \prec u_r \prec u_{r-1} \prec \cdots \prec u_1 \prec W.$$

Note that from these two types of voters, we add $2r-2$ to $M(W, u_i)$ for all $i \in [r]$, add 0 to $M(u_i, W)$ for all $i \in [r]$, and add $r-1$ to all other important edges. We can apply the same idea for other voters and manipulate the edge weights as follows.

- We can add $2r$ voters whose preference lists will add $2r$ to $M(W, W')$, add 0 to $M(W', W)$, and add r to all other edges. For this case, our construction is the same as the McGarvey's method [33].
- We can add $2r$ voters whose preference lists will add $2r$ to $M(W', v_j)$ for every $j \in [r]$, add 0 to $M(v_j, W')$ for every $j \in [r]$, and add r to all other edges.
- We can add $2r$ voters whose preference lists will add $2r$ to $M(v_j, W)$ for every $j \in [r]$, add 0 to $M(W, v_j)$ for every $j \in [r]$, and add r to all other edges.

Overall, the values $M(u, v)$ are summarized in Table 1, and the edge weights of the weighted majority graph are summarized in Table 2.

u \ v	W	W'	u_i	v_j
W	$\star$	$6r-1$	$6r-2$	$4r-1$
W'	$4r-1$	$\star$	$5r-1$	$6r-1$
u_i	$4r$	$5r-1$	$\star$	$2C_{i,j}+4r-1$
v_j	$6r-1$	$4r-1$	$6r-1-2C_{i,j}$	$\star$

Table 1. The values $M(u, v)$. The entries marked as $\star$ are not important in our analysis.

u \ v	W	W'	u_i	v_j
W	$\star$	$2r$	$2r-2$	$-2r$
W'	$-2r$	$\star$	0	$2r$
u_i	$-2r+2$	0	$\star$	$4C_{i,j}-2r$
v_j	$2r$	$-2r$	$2r-4C_{i,j}$	$\star$

Table 2. The weights $w(u,v)$ in the weighted majority graph. The entries marked as $\star$ are not important in our analysis.

Claim 6.3. *The Dominance Product C between A and B contains an entry of value r if and only if W is not a Schulze winner in the preference profile described above.*

Proof. Suppose C contains an entry $C_{i,j}$ where $C_{i,j} = r$ for some $i, j \in [r]$. In this case, $w(u_i, v_j) = 2r$ and $w(v_j, W) = 2r$, so $B_G(u_i, W) \geq 2r$. From the u_i column in Table 2 we have that all weights of edges that enter the set $\{u_1, \ldots, u_r\}$ are at most $2r-2$ (recall our assumption that all entries in C are positive). Therefore, any path going from W to u_i must use such an entering edge, so $B_G(W, u_i) \leq 2r-2 < B_G(u_i, W)$. Thus, W is not a winner.

Now we prove the reverse direction. Suppose C has no entry of value r. First of all, $B_G(W, W') \geq 2r$ since $w(W, W') = 2r$. To travel from W' to W, the last edge will have a value corresponding to the column W in Table 2, so the last edge has weight at most $2r$. Therefore, $B_G(W', W) \leq 2r \leq B_G(W, W')$.

Secondly, for any $j \in [r]$, $B_G(W, v_j) \geq 2r$ since the path $W \to W' \to v_j$ has bottleneck $2r$. Same as the previous case, in order to travel from v_j to W, the last edge has weight at most $2r$, so $B_G(v_j, W) \leq 2r \leq B_G(W, v_j)$.

Finally, for any $i \in [r]$, $B_G(W, u_i) \geq 2r-2$ since $w(W, u_i) = 2r-2$. To travel from u_i to W, we need to leave the set $\{u_1, \ldots, u_r\}$ at some point. When we do it, we use an important edge weight from the row u_i of Table 2. Since C has no weight r entry, all important weights in row u_i of Table 2 are at most $2r-4$. Therefore, $B_G(u_i, W) \leq 2r-4 < B_G(W, u_i)$.

Thus, W is a winner when C has no entry of value r. □

Hence, if we run the assumed $O(T(n)) = O(T(10r-2))$ time algorithm on the preference profile described above to check whether candidate W is a Schulze winner, we could use Claim 6.3 to decide if C has an entry of value r, and thus solve the Dominating Pairs problem. If $T(n)$ is super polynomial, then the theorem trivially holds as the Dominating Pairs problem is polynomial-time solvable; otherwise we have $O(T(10r-2)) = O(T(r))$. □

7 CONCLUSIONS

This paper considered the Schulze voting method and gave new algorithms and fine-grained conditional lower bounds for central problems such as computing the weighted majority graph (useful for other voting rules as well), computing all winners and verifying whether a candidate is a winner.

It is worth mentioning that while we focused on weighted majority graphs similarly to previous works[10] [12, 24, 25, 34–36] our algorithms work for *arbitrary* weighted directed graphs; let us call these *comparison graphs*. This means that we cover all possible weak orders $\succeq_D$ on $\mathbb{N}_0 \times \mathbb{N}_0$ that compare *the strength of the link* $(M(u,v), M(v,u)) \in \mathbb{N}_0 \times \mathbb{N}_0$ [40]. For a given instance with n voters and m candidates there are exactly $m(m-1)$ direct ordered comparisons between the

[10]These assumed additionally that votes are linear orders—in contrast, we allow weak preference orders.

candidates. Even if $\succeq_D$ is defined on $(n+1)^2$ different pairs, only at most $m(m-1)$ pairs appear in the instance. Therefore, for a given instance, we can encode the present pairs as numbers from the set $\{1, 2, \ldots, m(m-1)\}$ and use a standard comparison relation instead of $\succeq_D$. Then we simply use these numbers as weights in the comparison graph. In such a way, our algorithms work for all comparison relations mentioned by Schulze [40], i.e., *margin*, *ratio*, *winning votes*, and *losing votes*. Note that a weighted majority graph is defined as a comparison graph for the relation $\succeq_{\text{margin}}$ s.t. $(a, b) \succeq_{\text{margin}} (c, d)$ if and only if $a - b \geq c - d$.

Despite the fact that we have a lower bound on constructing the weighted majority graph it does not mean we have a lower bound on any voting rule which is defined using the weighted majority graph (e.g. tournament solutions [8]). Indeed, this does not exclude an other way of finding a winner under a voting rule without construction of the weighted majority graph. In the Schulze method it happens that the lower bound for a winner determination (Theorem 6.2) is almost the same (up to technical details) as the lower bound for constructing the weighted majority graph (Theorem 6.1). It is interesting to investigate for which voting rules (originally defined using the weighted majority graph) we can find winners in time faster than $O(m^{2.5-\varepsilon})$ for some $\varepsilon > 0$. It would require to use different techniques than these presented in the paper.

On the other side, it is worth applying a fine-grained complexity approach on voting rules similar to the Schulze method, in particular, these satisfying the axioms listed in the Section 1. Despite those similarities, they might require different techniques than used in this paper. More generally, it is interesting to research on fine-grained complexity for other computational social choice problems (not necessarily related to graph problems) that already have polynomial time algorithms (e.g., dynamic programming approach used for singled-peaked elections [7]).

ACKNOWLEDGMENTS

We thank David Eppstein for alerting us to the Wikipedia article [50]. Krzysztof Sornat was partially supported by the National Science Centre, Poland (NCN; grant number 2018/28/T/ST6/00366). Virginia Vassilevska Williams was supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship. Yinzhan Xu was supported by NSF Grant CCF-1528078.

REFERENCES

[1] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In *Proceedings of SODA 2021*. 522–539.

[2] Haris Aziz, Felix Brandt, Edith Elkind, and Piotr Skowron. 2019. Computational Social Choice: The First Ten Years and Beyond. In *Computing and Software Science - State of the Art and Perspectives*. LNCS, Vol. 10000. Springer, 48–65.

[3] Siddharth Barman, Omar Fawzi, and Paul Fermé. 2021. Tight Approximation Guarantees for Concave Coverage Problems. In *Proceedings of STACS 2021*. 9:1–9:17.

[4] Siddharth Barman, Omar Fawzi, Suprovat Ghoshal, and Emirhan Gürpinar. 2020. Tight Approximation Bounds for Maximum Multi-Coverage. In *Proceedings of IPCO 2020*. 66–77.

[5] Jan Behrens, Axel Kistner, Andreas Nitsche, and Björn Swierczek. 2014. *The Principles of LiquidFeedback*. Interaktive Demokratie e. V.

[6] Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. 2019. Decremental Strongly-Connected Components and Single-Source Reachability in Near-Linear Time. In *Proceedings of STOC 2019*. 365–376.

[7] Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. 2013. On the Computation of Fully Proportional Representation. *J. Artif. Intell. Res.* 47 (2013), 475–519.

[8] Felix Brandt, Markus Brill, and Paul Harrenstein. 2016. Tournament Solutions. In *Handbook of Computational Social Choice*. Cambridge University Press, 57–84.

[9] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia (Eds.). 2016. *Handbook of Computational Social Choice*. Cambridge University Press.

[10] Robert Bredereck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J Woeginger. 2014. Parameterized Algorithmics for Computational Social Choice: Nine Research Challenges. *Tsinghua Science and*

Technology 19, 4 (2014), 358–373.
[11] Markus Brill. 2018. Interactive Democracy. In *Proceedings of AAMAS 2018*. 1183–1187.
[12] Theresa Csar, Martin Lackner, and Reinhard Pichler. 2018. Computing the Schulze Method for Large-Scale Preference Data Sets. In *Proceedings of IJCAI 2018*. 180–187.
[13] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer.
[14] Britta Dorn and Ildikó Schlotter. 2017. Having a Hard Time? Explore Parameterized Complexity! In *Trends in Computational Social Choice*. AI Access, 209–230.
[15] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity*. Springer.
[16] Ran Duan and Seth Pettie. 2009. Fast Algorithms for (max, min)-Matrix Multiplication and Bottleneck Shortest Paths. In *Proceedings of SODA 2009*. 384–391.
[17] Szymon Dudycz, Pasin Manurangsi, Jan Marcinkowski, and Krzysztof Sornat. 2020. Tight Approximation for Proportional Approval Voting. In *Proceedings of IJCAI 2020*. 276–282.
[18] Edith Elkind, Piotr Faliszewski, and Arkadii M. Slinko. 2012. Clone Structures in Voters' Preferences. In *Proceedings of EC 2012*. 496–513.
[19] Ulle Endriss (Ed.). 2017. *Trends in Computational Social Choice*. AI Access.
[20] Shimon Even and Yossi Shiloach. 1981. An On-Line Edge-Deletion Problem. *J. ACM* 28, 1 (1981), 1–4.
[21] Piotr Faliszewski, Pasin Manurangsi, and Krzysztof Sornat. 2019. Approximation and Hardness of Shift-Bribery. In *Proceedings of AAAI 2019*. 1901–1908.
[22] Piotr Faliszewski and Rolf Niedermeier. 2016. Parameterization in Computational Social Choice. In *Encyclopedia of Algorithms*. 1516–1520.
[23] Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. 2020. A Survey on Approximation in Parameterized Complexity: Hardness and Algorithms. *Algorithms* 13, 6 (2020), 146.
[24] Serge Gaspers, Thomas Kalinowski, Nina Narodytska, and Toby Walsh. 2013. Coalitional Manipulation for Schulze's Rule. In *Proceedings of AAMAS 2013*. 431–438.
[25] Lane A. Hemaspaandra, Rahman Lavaee, and Curtis Menton. 2016. Schulze and Ranked-Pairs Voting are Fixed-Parameter Tractable to Bribe, Manipulate, and Control. *Ann. Math. Artif. Intell.* 77, 3-4 (2016), 191–223.
[26] Subhash Khot. 2002. On the Power of Unique 2-Prover 1-Round Games. In *Proceedings of STOC 2002*. 767–775.
[27] Karim Labib, Przemysław Uznański, and Daniel Wolleb-Graf. 2019. Hamming Distance Completeness. In *Proceedings of CPM 2019*. 14:1–14:17.
[28] François Le Gall. 2014. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of ISSAC 2014*. 296–303.
[29] François Le Gall and Florent Urrutia. 2018. Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor. In *Proceedings of SODA 2018*. 1029–1046.
[30] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. 2020. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In *Proceedings of ITCS 2020*. 53:1–53:18.
[31] Evangelos Markakis. 2017. Approximation Algorithms and Hardness Results for Fair Division with Indivisible Goods. In *Trends in Computational Social Choice*. AI Access, 231–247.
[32] Jiří Matoušek. 1991. Computing Dominances in E^n. *Inf. Process. Lett.* 38, 5 (1991), 277–278.
[33] David C. McGarvey. 1953. A Theorem on the Construction of Voting Paradoxes. *Econometrica* 21, 4 (1953), 608–610.
[34] Curtis Glen Menton and Preetjot Singh. 2013. Control Complexity of Schulze Voting. In *Proceedings of IJCAI 2013*. 286–292.
[35] David C. Parkes and Lirong Xia. 2012. A Complexity-of-Strategic-Behavior Comparison between Schulze's Rule and Ranked Pairs. In *Proceedings of AAAI 2012*. 1429–1435.
[36] Yannick Reisch, Jörg Rothe, and Lena Schend. 2014. The Margin of Victory in Schulze, Cup, and Copeland Elections: Complexity of the Regular and Exact Variants. In *Proceedings of STAIRS 2014*. 250–259.
[37] Aviad Rubinstein and Virginia Vassilevska Williams. 2019. SETH vs Approximation. *SIGACT News* 50, 4 (2019), 57–76.
[38] Donald G. Saari. 1994. *Geometry of Voting*. Springer, Berlin, Heidelberg.
[39] Markus Schulze. 2003. A New Monotonic and Clone-Independent Single-Winner Election Method. *Voting Matters* 17, 1 (2003), 9–19.
[40] Markus Schulze. 2011. A New Monotonic, Clone-Independent, Reversal Symmetric, and Condorcet-Consistent Single-Winner Election Method. *Soc. Choice Welf.* 36, 2 (2011), 267–303.
[41] Markus Schulze. 2018. The Schulze Method of Voting. *CoRR* abs/1804.02973 (2018).
[42] Asaf Shapira, Raphael Yuster, and Uri Zwick. 2011. All-Pairs Bottleneck Paths in Vertex Weighted Graphs. *Algorithmica* 59, 4 (2011), 621–633.
[43] Piotr Skowron, Piotr Faliszewski, and Jérôme Lang. 2016. Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation. *Artif. Intell.* 241 (2016), 191–216.
[44] John H. Smith. 1973. Aggregation of Preferences with Variable Electorate. *Econometrica* 41, 6 (1973), 1027–1041.

[45] Mikkel Thorup. 1999. Decremental Dynamic Connectivity. *J. Algorithms* 33, 2 (1999), 229–243.
[46] T. N. Tideman. 1987. Independence of Clones as a Criterion for Voting Rules. *Soc. Choice Welf.* 4, 3 (1987), 185–206.
[47] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. 2009. All Pairs Bottleneck Paths and Max-Min Matrix Products in Truly Subcubic Time. *Theory Comput.* 5, 1 (2009), 173–189.
[48] Virginia Vassilevska Williams. 2012. Multiplying Matrices Faster than Coppersmith-Winograd. In *Proceedings of STOC 2012*. 887–898.
[49] Virginia Vassilevska Williams. 2019. On Some Fine-Grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians ICM 2018*, Vol. 3. 3447–3487.
[50] Wikipedia contributors. 2021. Widest Path Problem—Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Widest_path_problem. [Online; accessed 26-May-2021].
[51] Raphael Yuster. 2009. Efficient Algorithms on Sets of Permutations, Dominance, and Real-Weighted APSP. In *Proceedings of SODA 2009*. 950–957.