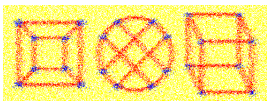
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 237–252 (2026)
DOI: 10.7155/jgaa.v30i1.3030

# Fast Schulze Voting Using Quickselect

*Arushi Arora*[1] *David Eppstein*[1] *Randy Le Huynh*[1]

[1]Computer Science Department, University of California, Irvine

Submitted: April 2025 Accepted: June 2026 Published: June 2026

Article type: Regular paper Communicated by: M. Zehavi

**Abstract.** The Schulze voting method aggregates voter preference data using maxmin-weight graph paths, achieving the Condorcet property that a candidate who would win every head-to-head contest will also win the overall election. Once the voter preferences among $m$ candidates have been arranged into an $m \times m$ matrix of pairwise election outcomes, a previous algorithm of Sornat, Vassilevska Williams and Xu (EC '21) determines the Schulze winner in randomized expected time $O(m^2 \log^4 m)$. We improve this to randomized expected time $O(m^2 \log m)$ using a modified version of quickselect.

## 1 Introduction

In ranked-choice voting, each voter provides a preference ordering of candidates rather than a single preferred candidate, and one of several methods may be used to aggregate these preferences and determine a winner. Prominent among these methods is one introduced by Markus Schulze in 2003 based on maxmin-weight paths in graphs, called the Schulze method or beatpath method. It has the advantage of being a *Condorcet method*: if one candidate would be preferred by a majority of voters in a head-to-head contest against each other candidate, that preferred candidate wins [10–12]. More generally, if a subset of candidates wins all head-to-head contests against other candidates, the Schulze winner will be in this subset. This property (the *Smith criterion* [13]) distinguishes the Schulze method from instant-runoff voting, the Borda count, and other commonly used ranked-choice methods [6].

In the Schulze method, voter preference orderings may be incomplete, specifying only a voter's preferences among a subset of candidates, or grouping some candidates together into equally-preferred subsets. From these votes, one can determine, for each pair of candidates, the number of voters who prefer one candidate over the other, and the *margin of victory* that the more-preferred candidate would hold over the less-preferred candidate in an election between only those two candidates. Schulze defines a weighted complete directed graph, in which the vertices are candidates, and an edge is directed from each pairwise winner to each pairwise loser, weighted by these margins

Research of David Eppstein was supported in part by NSF grant CCF-2212129.

*E-mail addresses:* arushia2@uci.edu (Arushi Arora) eppstein@uci.edu (David Eppstein) randylh@uci.edu (Randy Le Huynh)

of victory. In this graph, Schulze considers all pairwise maxmin-weight paths. The candidates can be partially ordered by the weights of these paths, considering one candidate to beat another when the maxmin weight from the first to the second is greater than the maxmin weight in the other direction. When the margins of victory are distinct, this partial order has a unique maximal element; the Schulze method chooses this candidate as the winner of the election, with a more complex tie-breaking procedure used in the case of a tie. A recent and frequently-updated preprint of Schulze surveys the usage of the Schulze method in governments, political parties, and organizations including the IEEE and ACM [12].

Determining the outcome of an election using the Schulze method has two separate algorithmic steps: aggregating individual ballots to convert voter preference orderings into a graph weighted by the margins of victory, and then determining a winner using maxmin-weight paths in this graph. In recent work, Sornat, Vassilevska Williams and Xu examined the fine-grained complexity of both subproblems, finding algorithms that (to within logarithmic factors) are optimal under standard complexity-theoretic assumptions [14]. Our work focuses on the second part, finding a winner from the weighted graph, for which we develop a simple, fast algorithm with fewer of those logarithmic factors.

The most obvious way to determine a Schulze winner would be to compute all-pairs maxmin-weight paths, use comparisons between path weights to order all candidates, and then search for a maximal element of the resulting partial order. The maxmin-weight path in a graph has been studied under many names, including the widest path, maximum capacity path, or bottleneck longest path. Maxmin-weight paths in undirected graphs are widely used for high-bandwidth network routing, and can be found between all pairs of vertices merely by finding a maximum spanning tree and using its paths. In directed graphs, the problem is not so simple, but it has also been well-studied and has multiple applications beyond voting. Many standard graph shortest path algorithms can be adapted to find maxmin-weight paths, and initial reference implementations of the Schulze method were based on the Floyd–Warshall algorithm for all-pairs shortest paths [10–12]. This textbook algorithm, published by Floyd in 1962 and closely related to earlier transitive closure algorithms by Roy and Warshall, takes time cubic in the number $m$ of candidates in the election [5, 9, 15].[1] Duan and Pettie [4] showed that all-pairs maxmin-weight paths can be computed by an algorithm based on fast matrix multiplication, taking time $O(m^{(3+\omega)/2})$ where $\omega$ is the exponent of fast matrix multiplication. As of 2026, $\omega \leq 2.37134$ giving a time bound for all pairs maxmin-weight paths of $O(m^{2.68567})$ [1].

However, as Sornat, Vassilevska Williams and Xu [14] observed, although computing all-pairs maxmin-weight paths is sufficient to determine the outcome of the Schulze method, it is not necessary. Instead, they found a faster algorithm for finding a Schulze winner (or winners, in case of ties). Their algorithm takes as input a weighted complete directed graph (representing the votes that would be cast for each candidate in each possible two-candidate election) and produces the Schulze method winner directly, in randomized expected time $O(m^2 \log^4 m)$. The algorithm does not use maxmin-weight paths at all, instead using a complex decremental strong connectivity data structure of Bernstein, Probst, and Wulff-Nilsen [2]. Our main result is a simpler, faster algorithm for determining the Schulze winner, taking randomized expected time $O(m^2 \log m)$. Our algorithm is an adaptation of another classical algorithm, quickselect [3, 7], using single-source maxmin-weight paths in a subroutine that replaces the pivoting steps in quickselect. The single logarithmic factor in our time bound comes from the fact that these maxmin-weight path subroutine calls consider the

[1] We follow Sornat, Vassilevska Williams and Xu [14] in using $m$ for the number of candidates and $n$ for the number of voters in an election. Thus, for the graph used in the Schulze method, the number of vertices is $m$, despite this variable more commonly being used for the edges in a graph.

whole given graph, even in later stages of the algorithm in which the field of candidates has been considerably narrowed down. As a subsidiary result, we show that this slowdown appears necessary for this algorithm: there exist inputs for which restricting these subroutine calls to a narrowed field of candidates would produce incorrect results. For the same reason, we cannot apply quicksort in place of quickselect to produce a near-quadratic-time Schulze ordering of all candidates.

Our $O(m^2 \log m)$ time bound is essentially optimal up to its single remaining logarithmic factor, as the input to our algorithm has size $O(m^2)$. We should note that under a lower bound (conditional with respect to standard assumptions of fine-grained complexity) proved by Sornat, Vassilevska Williams and Xu [14], the time for this step in determining a Schulze winner will be dominated by the time for converting voter preference orderings into a matrix of pairwise outcomes, which we do not speed up. Moreover, their lower bound holds for determining the Schulze winner from voter preference input, regardless of whether this conversion is made. Nevertheless, our improvement may have some practical applicability to other problems where the matrix of pairwise outcomes is already available, eliminating the need for conversion to a matrix. For example, it may be of interest to perform computational experiments on synthetic matrix data, or to analyze the outcome of a round-robin tournament for a high-scoring sport such as basketball by applying the Schulze method to the matrix of scores obtained by each team in each game.

## 1.1 New Results

We summarize here the new results of this paper:

- Given an $m \times m$ matrix of the margins of victory in head-to-head contests between each pair of $m$ candidates in an election (such as that produced by tallying these outcomes from ranked-choice voting data), we can find a Schulze method winner by a randomized algorithm whose time is $O(m^2 \log m)$ in expectation (Theorem 1) and with high probability (Theorem 2).

- For elections where multiple head-to-head contests have equal margins of victory, there may be multiple tied Schulze winners. In such cases, for an election with $k$ tied Schulze winners, we can list all winners in expected time $O(km^2 \log m)$ (Theorem 3).

- For elections where the margins of victory are all distinct, the partial order on candidates generated by the Schulze method is known to have a unique winner and a unique loser. We prove that, excluding these two candidates, the partial order on the remaining candidates can be arbitrary (Theorem 4), ruling out the possibility of using this method to rank all candidates.

- We provide examples of elections for which resolving ties (simultaneously) using a single random ballot cannot be used to generate a consistent partial ordering on all candidates (Example 1) and for which an attempt to speed up our algorithm by restricting its widest path calculations to the remaining non-eliminated candidates would produce incorrect results (Example 2).

# 2 Preliminaries

## 2.1 Top-Heavy Partial Orders and Quickselect

**Definition 1.** A *strict partial order* on a set $S$ is a binary relation $<$ on $S$ that obeys the following properties:

- Asymmetry: for every $x$ and $y$ in $S$, at most one of $x < y$ and $y < x$ is true. In particular, setting $y = x$, this implies irreflexivity: $x \not< x$.
- Transitivity: for every $x$, $y$, and $z$ in $S$, if $x \leq y$ and $y \leq z$, then $x \leq z$.

We say that a strict partial order is *top-heavy* if it has a unique top element $t$, such that for all $x \in S$, $x \neq t$ implies $x < t$. We say that two elements are *incomparable*, written as $x||y$, when neither $x < y$ nor $x > y$ is true.

A *linear extension* of a strict partial order is an arrangement of its elements into a sequence $x_0, x_1, \ldots x_{n-1}$ such that, whenever $x_i < x_j$ in the partial order, we have $i < j$ in the sequence. In computer science terms, it is a topological ordering of the directed acyclic graph with a directed edge from the smaller element to the larger element for each pair of comparable elements of the partial order.

Our algorithm will use a version of quickselect, specialized to find the top element of a top-heavy strict partial order. It can be defined by the following steps:

**ALGORITHM 1:** Quickselect for the top element of a top-heavy strict partial order

1. Choose a uniformly random element $p$ of the given set $S$ as a pivot.
2. By comparing each element to $p$, partition $S$ into the three subsets of elements $L = \{x : x < p\}$, $I = \{x : x||p\}$, and $H = \{x : p < x\}$.
3. If $H$ is non-empty, return the result of recursing into $H$.
4. Otherwise, $I$ must equal $\{p\}$, and $p$ must be the unique top element; return $p$.

Consider any fixed linear extension of $S$ (unknown to the algorithm). Then, in each pivoting step, the set $H$ of elements greater than the pivot $p$ must form a subset of the elements that follow $p$ in the linear extension. It follows that, regardless of the structure of the partial order $S$, the distribution of sizes of $H$ (depending on the random choice of pivot) is minorized by the distribution of sizes of the subsets that would be chosen when applying quickselect to a linear order. Thus, if we could perform constant-time comparisons (not true in our application), the time for this algorithm would be linear, regardless of the partial order. However, our algorithm for Schulze voting will use a modified form of quickselect in which the individual comparisons of the pivoting step are replaced by a maxmin-weight path computation. This replacement will have significant effects on runtime because it will involve maxmin-weight paths in the entire input graph rather than being restricted to a recursive subproblem.

## 2.2 The Schulze Method

Here we describe the Schulze method and some of its basic properties. We do not claim any originality for our observations about the method in this section. Although intended for use with preference ballots, and for the pairwise vote differentials in head-to-head contests between all pairs

of candidates, the Schulze method does not require all voters to specify their preference between all pairs of candidates.

**Definition 2.** For a given system of candidates $M$, voters $N$, and strict partial order voter preferences, let $P(x, y)$ denote the number of voters who prefer candidate $x$ to candidate $y$. Define the *weighted majority graph* to be a graph $G_{M,N,P}$ with $M$ as vertices, and with a directed edge from $x$ to $y$ for every pair $(x, y)$ of candidates with $x \neq y$. Label the edge from $x$ to $y$ with the weight $P(x, y) - P(y, x)$.

$G_{M,N,P}$ is a complete directed graph with antisymmetric integer weights, but we will not use these properties, and these are the only properties that it has. Every weighted graph with even weights can be realized as a graph $G_{M,N,P}$, for a large enough set $N$ of voters, even when requiring the voters to list a total order for all candidates (a requirement not made in the Schulze method), by adapting a method of McGarvey [8] for constructing preference systems that produce arbitrary patterns of pairwise outcomes. To do this, choose a parameter $k$ such that $2k$ is at least the largest pairwise margin of victory, and construct a pool of $N = 2k\binom{M}{2}$ voters. For each pair of candidates, choose $2k$ of these voters to prefer the two candidates as their top two choices, with the desired pairwise margin of victory. For $k$ of these voters, choose an arbitrary ordering of the remaining candidates, and for the other $k$ voters, use the reverse of the same ordering, so that the preferences for all other candidates cancel out. When voters may provide partial orders, achieving any system of weights is trivial, using a number of voters equal to the sum of weights, each of whom supplies a partial order that prefers one candidate over another and is indifferent to all other candidates.

**Definition 3.** For a weighted directed graph $G$ with a given set of candidates as its vertices, we define the *beatpath strength* $B(x, y)$ of candidate $x$ against candidate $y$ as the minimum weight of an edge on a path in $G$ from $x$ to $y$, if such a path exists, with the path chosen to maximize this minimum weight. If $x = y$ we define the beatpath strength to be $B(x, x) = +\infty$ and if $x \neq y$ and no path from $x$ to $y$ exists we define the beatpath strength to be $B(x, y) = -\infty$.

Define the *Schulze order* on the vertices of $G$ by the relation $<$, where for vertices $x$ and $y$ of $G$, we define $x < y$ whenever $B(x, y) < B(y, x)$.

The original version of Schulze's method [10] omits the negatively-weighted edges from $G_{M,N,P}$, but this merely complicates the definition of the Schulze order by making some values $B(x, y)$ undefined, without making any difference in the resulting Schulze order, as each two candidates have at least one non-negative path in one direction.

**Lemma 1** (Schulze [11], Section 4.1)**.** *The Schulze order is a strict partial order.*

**Lemma 2** (Schulze [11], Section 4.2.1)**.** *If the weights of the edges of a graph $G$ are all distinct, the Schulze order is top-heavy.*

When the Schulze order is top-heavy, the top element of the order is declared to be the Schulze winner. Sornat, Vassilevska Williams and Xu [14] describe an algorithm for finding this Schulze winner in expected time $O(m^2 \log^4 m)$, based on the deletion of edges from the graph $G_{M,N,P}$ in ascending order by weight and the use of a data structure for decremental strongly connected components in dynamic graphs.

When the edge weights are not distinct, there can be multiple maximal elements in the Schulze order. In such cases, Schulze [10–12]. proposes breaking the tie by using the preference ordering of a randomly chosen voter. Rather than implementing this tie-breaking method directly, Sornat, Vassilevska Williams and Xu [14] modify their decremental strongly connected component algorithm to find the set of all maximal elements in the same expected time, $O(m^2 \log^4 m)$.

# 3 Algorithm and its Complexity Analysis

We leverage quickselect to identify a Schulze winner without the need of an intermediate step of computing all pairs maxmin-weight paths, and without using decremental strongly connected components.

## 3.1 Subroutines

Although we do not use all pairs maxmin-weight paths, we do nevertheless compute some maxmin-weight paths in the graph $G_{M,N,P}$. For this we use a fast version of Dijkstra's algorithm optimized for dense graphs.

**Lemma 3.** *Single-source or single-destination maxmin-weight paths in a graph with $m$ vertices can be computed for a single designated source or destination vertex in time $O(m^2)$.*

**Proof:** For single-source maxmin-weight paths, we use Dijkstra's algorithm (with the minimum weight of an edge on a path as its priority rather than the sum of weights), without any priority queue data structure. Instead, we merely maintain the maximum priority found so far for a path to each unprocessed vertex, and in each step choose the next vertex $u$ to process by scanning all unprocessed vertices sequentially and choosing the one whose priority is maximum. Then, as is usual for Dijkstra's algorithm, we consider the paths formed by following one more edge $uv$ from the chosen vertex to each unprocessed vertex $v$, we compute the priority of each such path as the minimum of the priority of $u$ and the weight of edge $uv$, and we update the priority of $v$ to the maximum of its old value and the priority of the path. In pseudocode:

**ALGORITHM 2:** Dense single-source maxmin-weight paths

1. Set the priority of every vertex to $-\infty$, and the priority of the source vertex to $+\infty$.
2. Flag each vertex as unprocessed, and initialize a list $U$ of unprocessed vertices.
3. While $U$ is non-empty:
   (a) Scan $U$ to find the maximum-priority vertex $u$; let its priority be $\beta$ (the bottleneck weight of the path from the source to $u$).
   (b) For each edge $uv$ with weight $w$, where $v$ is unprocessed, set the priority of $v$ to the maximum of its old priority with $\min(\beta, w)$.
   (c) Remove $u$ from $U$

There are $m$ vertices to process, finding the next vertex to process takes time $O(m)$, and processing each vertex takes time $O(m)$, so the total time is $O(m^2)$ as claimed. For single-destination maxmin-weight paths, we apply the same algorithm to the graph obtained by reversing all edges in the given graph. □

Our main idea is that, when a vertex $p$ is selected as the pivot in quickselect, we can use two instances of this single-source or single-destination path computation to perform all comparisons in the Schulze method with respect to $p$.

**Lemma 4.** *Let $p$ be any vertex of $G_{M,N,P}$, and let $S$ be any set of vertices. Then in time $O(m^2)$ we can determine, for each vertex $v$ in $S$, whether $v < p$, $v||p$, or $p < v$.*

**Proof:** We use the following algorithm:

**ALGORITHM 3:** Pivoting on a single vertex $p$

1. Apply Algorithm 2 to $G_{M,N,P}$, using a single-source computation from $p$, to compute each beatpath weight $B(p, v)$.
2. Apply Algorithm 2 to $G_{M,N,P}$ again, using a single-destination computation to $p$, to compute each beatpath weight $B(v, p)$.
3. Partition the candidates into the three sets $L = \{v : B(v, p) < B(p, v)\}$, $I = \{v : B(v, p) = B(p, v)\}$, and $H = \{v : B(v, p) > B(p, v)\}$, and return these three sets.

The fact that these three sets are the sets of elements less than, incomparable to, and greater than the pivot $p$ in the Schulze order follows immediately from the definition of the order. The time bound follows from Lemma 3. □

## 3.2 The Main Algorithm

Our main algorithm adapts our quickselect-based algorithm for a maximal element in any top-heavy partial order (Algorithm 4) to use this pivoting algorithm, and then checks for uniqueness. It is convenient to use an iterative version of quickselect rather than a recursive version, but this makes little difference to the algorithm and its correctness.

**ALGORITHM 4:** Quickselect for the top element of the Schulze order

1. Let $S$ be the set of all vertices in $G_{M,N,P}$.
2. While $|S| > 1$, do the following steps:
   (a) Choose a uniformly random element $p$ of the given set $S$ as a pivot.
   (b) Use Algorithm 3 to partition $S$ into the three subsets of elements $L = \{x : x < p\}$, $I = \{x : x||p\}$, and $H = \{x : p < x\}$.
   (c) If $H$ is non-empty, set $S = H$; otherwise, set $S = \{p\}$.
3. Let $p$ be the unique member of $S$, guaranteed to be a maximal element of the Schulze order.
4. To test whether $p$ is the unique maximal element, use Lemma 4 again to partition the vertices of $G_{M,N,P}$ into the three subsets of elements $L = \{x : x < p\}$, $I = \{x : x||p\}$, and $H = \{x : p < x\}$. $H$ is guaranteed to be empty; the order is top-heavy with $p$ as the unique winner if and only if $I$ is a singleton set.

In the case that the order is not top-heavy, the maximal elements of the Schulze order are exactly the maximal elements of its restriction to the final set $I$ from step 4 of the algorithm. Our algorithm does not determine which elements of $I$ are maximal. There must be at least two of these maximal elements, $p$ and any maximal element of $I \setminus \{p\}$. It is necessary to recompute $I$ in this step rather than re-using the last such set computed in step 2(b), because this final set may include elements discarded in earlier iterations of the algorithm.

**Lemma 5.** *Algorithm 4 finds a maximal element of the Schulze order and correctly determines whether it is the unique maximal element.*

**Proof:** By induction on the number of steps of the while loop, in each step the sequence of already-chosen pivots are all comparable in the Schulze order, and $S$ consists of the elements that

are greater than all of these pivots. Thus, in the final iteration of the loop, when $H$ becomes empty, there are no elements greater than the final chosen pivot $p$, so $p$ is maximal.

If $p$ is the unique maximal element, then there can be no other element greater than it or incomparable to it, the sets $H$ and $I$ found in step 4 will be empty and the singleton $\{p\}$ respectively, and the algorithm will correctly report that $p$ is unique. In the other direction, if $p$ is maximal but not the unique maximal element, then any other maximal element will be incomparable with $p$, it will be included in the set $I$ found in step 4, and the algorithm will correctly report that $p$ is not the unique maximal element. □

In Algorithm 4, the main contributor to the total time is the call to Algorithm 3 in the inner loop, which takes time $O(m^2)$ according to Lemma 4. Thus, to analyze the algorithm we mainly need to determine how many times this loop iterates. In the next two subsections we determine both the expected runtime of Algorithm 4 and high-probability bounds on its runtime, showing a time of $O(m^2 \log m)$ in both cases.

## 3.3 Expected Time

In both this and the next section our analysis of Algorithm 4 is more or less the standard analysis of the usual quickselect algorithm, with the exception that our emphasis is on the number of rounds of iteration made by the algorithm rather than its number of comparisons. This change of emphasis corresponds to the fact that each round takes complexity linear in the size of the entire weighted majority graph, rather than (as in standard quickselect) becoming faster as the number of remaining candidates becomes smaller.

Let $T(m)$ denote the expected runtime of Algorithm 4 on a worst-case input of size $m$, and let $R(m)$ denote the expected number of iterations of the while loop of the algorithm, again on a worst-case input of size $m$. Each iteration takes time $O(m^2)$ by Lemma 4, and the work performed outside of the while loop is also $O(m^2)$ for the same reason; thus, $T(m) = O(m^2 R(m))$. In the analysis of this section, we show that $R(m) = O(\log m)$ and therefore that $T(m) = O(m^2 \log m)$. This would also follow from our high-probability analysis but for expected time we can obtain more precise bounds on the number of iterations.

**Lemma 6.** *Let $S$ have size $s$ before the start of an iteration of the while loop of the algorithm, and let $S'$ be the new value of $S$ after the iteration. Then, for each possible size $i$ of $S'$ (with $1 \le i < s$),*

$$\Pr[|S'| \le i] \ge \frac{(i+1)}{s}.$$

**Proof:** Fix any linear extension of the Schulze order, and number the elements of $S$ as $x_1, x_2, \ldots x_s$ from larger to smaller in this extension ordering. Then if pivot $x_j$ is chosen, then either $S' = \{x_j\}$ (with size one) or $S'$ contains only (a subset of the) elements $x_k$ with $k < j$ (with size at most $j-1$). Thus, whenever $j \le i+1$ the size of $S'$ will be at most $i$, and this choice of $j$ happens with probability $(i+1)/s$. □

**Corollary 1.** *$R(m)$ obeys the recurrence*

$$R(m) = 1 + \frac{1}{m} \sum_{i=1}^{m} R(i-1)$$

*with base case $R(0) = R(1) = 0$.*

**Proof:** The base case follows as while loop will immediately stop iterating when $S$ has fewer than two elements. Because $R(m)$ is obviously monotonic in $m$, the worst case size distribution for the size after each iteration, obeying Lemma 6, is the distribution in which size 1 has probability $2/s$ and each other size has probability $1/s$. The recurrence of the corollary describes exactly this worst-case size distribution after the trivial substitution of size 0 and 1 with probability $1/s$ each in place of size 1 with probability $2/s$.

We have equality for the recurrence rather than inequality, because this worst case distribution can be achieved, for each iteration of the algorithm, when the input Schulze ordering is a linear ordering. For instance this would be true when there is a single voter with that ordering as their preference. □

**Theorem 1.** *$R(m) = \ln m + O(1)$ and $T(m) = O(m^2 \log m)$.*

**Proof:** Let $\rho(m)$ obey the same recurrence as Corollary 1, with a changed base case $\rho(0) = 0$, $\rho(1) = 1$. Then by induction on $m$, $\rho(m) = \sum_{i=1}^{m} \frac{1}{i}$, a harmonic number. To see this, use the induction hypothesis to expand and regroup

$$\begin{aligned}
\rho(m) &= 1 + \frac{1}{m} \sum_{i=1}^{m-1} \sum_{j=1}^{i} \frac{1}{j} \\
&= 1 + \frac{1}{m} \left( 1 + \sum_{j=2}^{m-1} \left( 1 + (m-j)\frac{1}{j} \right) \right) \\
&= 1 + \frac{1}{m} \left( 1 + \sum_{j=2}^{m-1} \frac{m}{j} \right) \\
&= 1 + \frac{1}{m} + \sum_{j=2}^{m-1} \frac{1}{j}.
\end{aligned}$$

As is well known, the $m$th harmonic number is $\ln m + O(1)$. The change of the base case decreases the overall value of $R(m)$ from the harmonic numbers but the amount of decrease is less than the change by one unit that would occur when decreasing both base cases by one. □

## 3.4 High Probability Analysis

Define an iteration of the while loop of Algorithm 4 to be a *halving iteration* if, after the iteration, $S$ has decreased to half its former size or smaller. Obviously, the algorithm will terminate after at most $\log_2 m$ halving iterations. By Lemma 6, each iteration has probability at least $1/2$ of being a halving iteration, independently of all previous iterations. We may thus model the process of Algorithm 4 by a sequence of random coin flips, stopping when $\log_2 m$ heads are reached. The actual algorithm may stop earlier (because the algorithm can terminate with fewer than $\log_2 m$ halving iterations) but any valid high-probability upper bound on the number of coin flips until stopping will be a valid bound on the number of iterations of the algorithm.

**Theorem 2.** *For every $c > 0$ there exists $\kappa > 0$ such that with probability $\geq 1 - \frac{1}{m^c}$ the algorithm terminates after at most $\kappa \log_2 m$ iterations. Thus, with high probability (polynomially close to one) it takes time $O(m^2 \log m)$.*

**Proof:** We use a standard Chernoff bound, in the form that for a sum $X$ of random 0-1 variables with expected value $\mu$,

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$$

for any chosen parameter value $\delta > 0$.

Here, we let $X$ be the number of *tails* of the random coin-flip process discussed above, in which we aim to get at least $\log_2 m$ heads. If the process consists of $\kappa \log_2 m$ trials, for a parameter $\kappa$ to be determined, we can calculate the expected number of tails as $\mu = \frac{1}{2}\kappa \log_2 m$. The process fails to get enough coin flips when

$$X > (\kappa - 1)\log_2 m = (1+\delta)\mu,$$

where to achieve the equality in this equation we set $\delta = 1 - 2/\kappa$.

For large values of $\kappa$, we will have $\delta$ close to one, and failure probability upper-bounded by something close to $(e/4)^\mu$. For instance, whenever $\kappa \geq 10$ we will have

$$\frac{e^\delta}{(1+\delta)^{1+\delta}} < 0.7726$$

and failure probability upper-bounded by

$$0.7726^{\frac{1}{2}\kappa \log_2 m} = m^{(\frac{1}{2}\kappa)\log_2 0.7726} < m^{-0.186\kappa}.$$

By setting $\kappa = \max\{10, c/0.186\}$ we can make the failure probability be smaller than any polynomial bound $1/m^c$, as desired. □

## 3.5 All Maximal Elements

Although multiple maximal elements are unlikely by Lemma 2, they can occur when multiple edges of the weighted majority graph have equal weight. As an extreme case, all weights can be zero causing all candidates to be tied. Algorithm 4 will detect these situations but will not list all maximal elements of the Schulze order; the set of elements incomparable with its chosen maximal element may be a strict superset of the maximal elements. Sornat, Vassilevska Williams and Xu describe how to modify their algorithm to obtain all maximal elements in randomized expected time $O(m^2 \log^4 m)$, the same as their time bound for finding a single maximal element [14]. While we do not achieve the same $O(m^2 \log m)$ time bound as we do for Algorithm 4, we can find all maximal elements more quickly when (as is likely) there are few of them:

**Theorem 3.** *It is possible to find all maximal elements of the Schulze order in expected time $O(km^2 \log m)$, where $k$ is the number of maximal elements.*

**Proof:** Apply the following algorithm.

**ALGORITHM 5:** All maximal elements of the Schulze order

1. Set $C$ to be the set of all candidates.
2. While $C$ is not empty:
   (a) Use Algorithm 4, with step 1 modified to set $S = C$ rather than setting it to the set of all vertices, to find a maximal element $p$ of $C$, and output $p$.
   (b) Use Lemma 4 to partition $C$ into the three subsets of elements $L = \{x : x < p\}$, $I = \{x : x||p\}$, and $H = \{x : p < x\}$. By the maximality of $p$, $H$ will be empty.
   (c) Set $C$ to $I \setminus \{p\}$, the subset of elements not less than or equal to $p$.

By induction, the set $C$ remaining after each iteration of the outer loop is the subset of candidates that are not less than or equal to any of the maximal elements found so far. Therefore, any maximal element in $C$ is a maximal element of the whole Schulze order, and all maximal elements will have been found when $C$ becomes empty. Each iteration of the while loop produces a single maximal element and takes time $O(m^2 \log m)$ by the analysis of Algorithm 4, which remains valid for the modified initial choice of $S$. □

# 4 Barriers to Improvement

Several natural directions for extension, generalization, or speedup of our algorithm are blocked by the counterexamples that we present in this section.

## 4.1 Tiebreak Order

Ties can occur in the Schulze method when two head-to-head contests have equal margins of victory (Lemma 2), creating the need for a tie-breaking method. Schulze [10–12]. suggests several tie-breaking methods, including the use of a randomly-drawn ballot (or sequence of ballots) to determine an ordering among the maximal elements of the Schulze order, without changing the order relations already determined by this order. One way to perform a random ballot based tie-breaking method would be to use Theorem 3 to list all maximal candidates, and then applying random balloting to this list. However, compared to using Algorithm 4, this would incur a time penalty proportional to the number of candidates listed. It is natural to hope that the winner after tie-breaking could be obtained more quickly by modifying the faster algorithm of Algorithm 4 or its input to use the additional information from a random ballot. However, as the example below shows, the result of any such modification cannot replicate the same comparisons between candidates that would be obtained by breaking all ties simultaneously using a single random ballot, because these comparisons do not necessarily form a consistent partial order.

**Example 1.** Consider an election with three candidates $A$, $B$, and $C$, and with four voters with preferences $A > B > C$, $C > A > B$, $A > C > B$, and $B > C > A$. Then in the weighted majority graph, shown in Figure 1, all edges into and out of $C$ have weight zero, while the edge from $A$ to $B$ has weight 2 and its reverse has weight $-2$. The beatpath strengths are the same except that the beatpath strength from $B$ to $A$ is zero (by a path through $C$), shown in Table 1. Thus, $A$ and $C$ are the two maximal elements. The only comparable pair in the Schulze order is $A > B$; the other two pairs are incomparable.

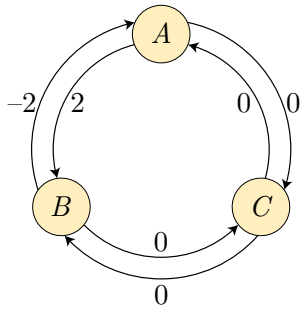


Figure 1: Weighted Majority Graph for Example 1

| | A | B | C |
|---|---|---|---|
| A | — | 2 | 0 |
| B | 0 | — | 0 |
| C | 0 | 0 | — |

Table 1: Beatpaths for Example 1

Now, suppose that the randomly chosen ballot is the one with the order $B > C > A$. We cannot determine a partial order by using this order as a tiebreaker between each pair of incomparable elements of the Schulze order, because this would produce the comparisons $A > B$ (not a broken tie), $B > C$ (by the tiebreak rule), and $C > A$ (by the tiebreak rule), giving a cyclic sequence of comparisons that is not allowed in a strict partial order.

Because this random ballot tiebreaking method does not produce a partial order that can be interpreted as a Schulze order for a perturbed ballot count, it may make sense to instead break ties by perturbing the weighted majority graph: randomly order the edges of this graph and, for an edge in position $i$ of the order, add $i/m^2$ to its weight. These perturbations are small enough that they cannot change the ordering among comparable pairs of candidates in the unperturbed Schulze ordering; they can only make an incomparable pair become comparable. All candidates are treated symmetrically by this perturbation method. It causes all edges to have distinct weights, from which by Lemma 2 there can be only one Schulze winner, the unique maximal element of the Schulze order. This winner can be found in $O(m^2 \log m)$ expected time by Algorithm 4.

## 4.2 Paths in Induced Subgraphs

Algorithm 4 takes time $O(m^2 \log m)$, rather than $O(m^2)$ (linear in the size of its input) because in each of the $O(\log m)$ iterations of its outer loop it applies a linear-time subroutine on the entire weighted majority graph. In contrast, the standard quickselect algorithm takes linear time, with the same number of iterations (or recursive calls), because later iterations with fewer elements take less time. It is natural to hope that we could obtain a similar speedup in Algorithm 4 by considering only maxmin-weight paths in a smaller subgraph, such as the induced subgraph of the previous pivot and the remaining candidates. As the next example shows, however, restricting to the induced subgraph in this way would produce incorrect results.

**Example 2.** Consider a weighted majority graph with edge weights shown in Figure 2.

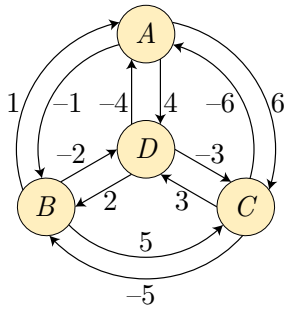


Figure 2: Weighted Majority Graph for Example 2

Calculating maxmin-weight paths between each two candidates produces the table of beatpath strengths shown below in Table 2 (left).

| | A | B | C | D |
|---|---|---|---|---|
| A | — | 2 | 6 | 4 |
| B | 1 | — | 5 | 3 |
| C | 1 | 2 | — | 3 |
| D | 1 | 2 | 2 | — |

| | A | B | C |
|---|---|---|---|
| A | — | −1 | 6 |
| B | 1 | — | 5 |
| C | −5 | −5 | — |

| | A | B |
|---|---|---|
| A | — | −1 |
| B | 1 | — |

Table 2: Beatpaths for Example 2 in the full graph, and in the subgraphs induced by $\{A, B, C\}$ and $\{A, B\}$

These strengths produce the Schulze ordering $A > B > C > D$ (a total order), in which $A$ is the unique maximal element and the unique Schulze winner.

Now suppose that Algorithm 4 selects $C$ as its first random pivot. If we modified the algorithm to compute subsequent paths in the induced subgraph of $\{A, B, C\}$ or of $\{A, B\}$, we would still have a path from $B$ to $A$ with maxmin weight 1; however, the removal of $D$ has eliminated all positive paths from $A$ to $B$, leaving the edge from $A$ to $B$ as the maxmin-weight path, with weight $-1$. Thus, the Schulze order for either induced subgraph would have $B > A$, different from the Schulze order of the whole graph. The modified algorithm that finds paths in either induced subgraph would produce an incorrect result.

## 4.3 Structure in the Schulze Order

Given our use of quickselect to find a maximal element in the Schulze order, it is natural to hope that a similar variation of quickselect might be used to find elements in intermediate positions in the order, or that a variation of quicksort might be used to determine the entire order. These hopes would be boosted if, for instance, the Schulze order were to turn out to be a weak order, a partition of the candidates into tied sets with a total ordering on those tied sets. Weak orders are commonplace in applications of the standard quicksort and quickselect algorithms (for which the input might be a collection of records with numerical priorities that can be tied) and the algorithms are easily adapted to this case.

Some structure in the Schulze order is provided by Lemma 2, according to which (for graphs with distinct edge weights) there is a unique maximal element. The same lemma, applied to an

edge-reversed version of the same graph, shows also that their Schulze orders have a unique minimal element. However, that is the only structure available in these orders, because every partial order with unique maximal and minimal elements can be realized in this way:

**Theorem 4.** *Let $M$ be a set of candidates with a strict partial order $<$ for which there is a unique maximal element $t$ and a unique minimal element $b$. Then there exist voters $N$ and preferences $P$ for which the Schulze order of the weighted majority graph $G_{M,N,P}$ is exactly the given strict partial order.*

**Proof:** $G_{M,N,P}$ must be a complete directed graph on the given candidate set $M$; it remains to assign weights to its edges in order to make the given order be the Schulze order. To do so, we use the following construction:

**Example 3.** Form pools of distinct positive weights to assign to the edges, which we call *large*, *medium*, *small*, and *tiny*, with all weights ordered as large > medium > small > tiny, and with enough distinct weights in each pool to assign to each edge of $G_{M,N,P}$. We will assign these weights in such a way that the comparisons between pairs of edges in the same pool as each other will not affect the resulting Schulze order. For each candidate $x$ or pair of candidates $x$ and $y$ in $M \setminus \{t, b\}$, we assign these weights to the edges of $G_{M,N,P}$ as follows:

- We assign a large weight to each edge from $t$ to $x$, and to each edge from $x$ to $b$ (and the negation of the same weight to the opposite edge).
- If $x > y$, we assign a medium weight to the edge from $x$ to $y$, and the negation of the same weight to the opposite edge.
- We assign a small weight to the edge from $b$ to $t$, and its negation to the opposite edge.
- If $x||y$, we assign a tiny weight to one of the two edges between $x$ and $y$, and its negation to the opposite edge.

With these weights, $t$ has a large-weight path to every other candidate, and the only positive incoming edge to $t$ has small weight, so $t$ is the (unique by Lemma 2) maximal element of the Schulze order, matching the given partial order. Symmetrically, $b$ has a large-weight path from every other element, and the only positive outgoing edge from $b$ has small weight, so $b$ is the unique minimal element of the Schulze order, matching the given partial order.

Each remaining pair of candidates $x$ and $y$ can be connected by a path $x$–$b$–$t$–$y$, in which the minimum weight edge is the edge from $b$ to $t$ with small weight. There is no large-weight path between them, and a medium-weight path exists if and only if $x > y$. Thus, the maxmin-weight path has medium weight if $x > y$, and otherwise it has small weight equal to the weight of the edge from $b$ to $t$. When $x > y$ in the given order, this gives $x > y$ in the Schulze order. When $x||y$ in the given order, this gives equal weight to the maxmin-weight paths from $x$ to $y$ and from $y$ to $x$, giving $x||y$ in the Schulze order. Thus in all cases the Schulze order matches the given order, as stated. □

# 5 Conclusions

We have presented a novel algorithm for computing a Schulze winner with an improved expected time complexity of $O(m^2 \log m)$, given as input a weighted majority graph, and a variant of the algorithm that finds all maximal elements of the Schulze order (in case of ties) in expected time $O(km^2 \log m)$, compared to a previous expected $O(m^2 \log^4 m)$ time bound for both problems [14].

As our algorithm is randomized (like the previous algorithm) it is natural to ask for the best time bound of a deterministic algorithm for the Schulze method. The previous algorithm of Sornat et al. [14] depends on randomized methods for decremental strong connectivity, for which all known deterministic methods are slower by a factor of $m$ [2]. Instead, the fastest deterministic time bounds that we are aware of for the Schulze method are based on computing all pairs widest paths using fast matrix multiplication [4]. Plugging in known bounds for matrix multiplication [1] gives a time bound of $O(m^{2.68567})$. The same time bound applies to finding the entire Schulze partial order, which neither our algorithm nor the algorithm of Sornat et al. [14] do. Additionally, although the examples in Section 4 provide barriers to certain natural directions in which our algorithm might be improved, they still leave open the possibility that an entirely different algorithm might achieve a faster time bound.

## References

[1] J. Alman, R. Duan, V. Vassilevska Williams, Y. Xu, Z. Xu, and R. Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*. SIAM, 2025. doi:10.1137/1.9781611978322.63.

[2] A. Bernstein, M. Probst, and C. Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In M. Charikar and E. Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.

[3] L. Devroye. Exponential bounds for the running time of a selection algorithm. *J. Comput. System Sci.*, 29(1):1–7, 1984. doi:10.1016/0022-0000(84)90009-6.

[4] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In C. Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 384–391. SIAM, 2009. doi:10.1137/1.9781611973068.43.

[5] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962. doi:10.1145/367766.368168.

[6] B. Grofman. Alternative voting methods. In C. Rowley and F. Schneider, editors, *The Encyclopedia of Public Choice*, pages 333–336. Springer US, 2004. doi:10.1007/978-0-306-47828-4_31.

[7] C. A. R. Hoare. Algorithm 65: Find. *Communications of the ACM*, 4(7):321–322, July 1961. doi:10.1145/366622.366647.

[8] D. C. McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, 21:608–610, 1953. doi:10.2307/1907926.

[9] B. Roy. Transitivité et connexité. *C. R. Acad. Sci. Paris*, 249:216–218, 1959.

[10] M. Schulze. A new monotonic and clone-independent single-winner election method. *Voting Matters*, 17:9–19, 2003. URL: https://www.mcdougall.org.uk/VM/ISSUE17/I17P3.PDF.

[11] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011. doi:10.1007/s00355-010-0475-4.

[12] M. Schulze. The Schulze Method of Voting. Electronic preprint arxiv:1804.02973, March 2018 – October 2025.

[13] J. H. Smith. Aggregation of preferences with variable electorate. *Econometrica*, 41(6):1027–1041, 1973. doi:10.2307/1914033.

[14] K. Sornat, V. Vassilevska Williams, and Y. Xu. Fine-grained complexity and algorithms for the Schulze voting method. In *Proceedings of the 22nd ACM Conference on Economics and Computation (EC '21), Budapest, Hungary*, pages 841–859. Association for Computing Machinery, 2021. doi:10.1145/3465456.3467630.

[15] S. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9:11–12, 1962. doi:10.1145/321105.321107.