Computing the Schulze Method for Large-Scale Preference Data Sets

Theresa Csar and **Martin Lackner** and **Reinhard Pichler**
TU Wien, Austria
{csar, lackner, pichler}@dbai.tuwien.ac.at

Abstract

The Schulze method is a voting rule widely used in practice and enjoys many positive axiomatic properties. While it is computable in polynomial time, its straight-forward implementation does not scale well for large elections. In this paper, we develop a highly optimised algorithm for computing the Schulze method with Pregel, a framework for massively parallel computation of graph problems, and demonstrate its applicability for large preference data sets. In addition, our theoretic analysis shows that the Schulze method is indeed particularly well-suited for parallel computation, in stark contrast to the related ranked pairs method. More precisely we show that winner determination subject to the Schulze method is NL-complete, whereas this problem is P-complete for the ranked pairs method.

1 Introduction

Preference aggregation is a core problem of computational social choice: how to aggregate potentially conflicting preferences of agents into a collective preference ranking or to identify most preferred alternatives. This problem occurs both in a technical context where agents correspond to machines or programs (e.g., multi-agent systems, group recommendation system, aggregation of information sources), and in a social context where people strive for a joint decision (e.g., political voting, group decision making). The Schulze method [Schulze, 2003; 2011] is a preference aggregation method—or *voting rule*—with a particularly attractive set of properties[1]. For example, the Schulze method satisfies the independence of clones criterion, i.e., if a candidate joins that is virtually indistinguishable from another candidate (i.e., a "clone" joins), then the relative ranking of alternatives in the output remains unchanged. Furthermore, the Schulze method is Condorcet-consistent and monotonic [Schulze, 2011]. Partially due to these advantages, the Schulze method enjoys sig-

nificant popularity and is widely used in group decisions of open-source projects and political groups[2].

Preference aggregation is often a computationally challenging problem. It is therefore fortunate that the Schulze method can be computed in polynomial time. More precisely, given preference rankings of n voters over m candidates, it requires $\mathcal{O}(nm^2 + m^3)$ time to compute the output ranking of all m candidates. However, while this runtime is clearly feasible for decision making in small- or medium sized groups, it does not scale well for a larger number of candidates due to the cubic exponent. Instances with a huge number of alternatives occur in particular in technical settings, such as preference data collected by e-commerce applications or sensor systems. If the number of alternatives is in the thousands, the classical algorithm for computing the Schulze method (based on the Floyd–Warshall algorithm) quickly reaches its limits.

The goal of our paper is to design a fast, parallel algorithm that enables preference aggregation via the Schulze method on large-scale data sets. As a first step, we perform a worst-case complexity analysis and show that the Schulze Method is indeed suitable for parallel computation: we prove that the problem of computing a Schulze winner (i.e., the top-ranked alternative) is NL-complete. We contrast this result by showing that the related ranked pairs method, which has similar axiomatic properties, is likely not to allow for effective parallel computation: here, we show that the winner determination problem is P-complete.

Building on this theoretic foundation, we design a parallel algorithm for computing Schulze winners in the Pregel framework. Pregel [Malewicz *et al.*, 2010] is a framework for cloud-based computation of graph problems. In Pregel, parallelization happens on the level of vertices, i.e., each vertex constitutes an independent computation unit that communicates with neighbouring vertices. The Schulze method is based on the weighted majority graph, i.e., it does not require actual rankings as input but rather pairwise majority margins for each pair of vertices. Hence, computing the Schulze method is a problem very suitable for the Pregel framework.

As the main contribution of this paper, we present a highly optimised Pregel-based parallel algorithm for computing Schulze winners. This algorithm can easily be adapted

---

[1]We note, however, that other voting rules satisfy similar or even stronger axiomatic properties, for example those defined by the minimal covering set, the uncovered set, or the bipartisan set [Brandt *et al.*, 2016].

to also computing the top-k alternatives. We demonstrate the effectiveness of our optimisations in an experimental evaluation. We use daily music charts provided by the Spotify application to generate data sets with up to 18,400 alternatives; the corresponding weighted tournament graphs have up to 160 million weighted edges. We show that such data sets can be computed in the matter of minutes and demonstrate that runtimes can be significantly reduced by an increase in parallelization. Thus, our algorithm enables the application of the Schulze method in data-intensive settings.

Structure and main results. We recall some basic notions and results in Section 2. A conclusion and discussion of future directions for research are given in Section 6. Our main results, which are detailed in Sections 3 – 5, are as follows:

- In Section 3 we carry out a complexity-theoretic analysis of winner determination for the Schulze method and for the ranked pairs method. We thus establish a significant difference between the two methods in that the former is in NL whereas the latter is P-hard. This explains why the Schulze method is very well-suited for parallelization.

- In Section 4, we present a new parallel algorithm for the winner determination according to the Schulze method, using the vertex-centric algorithmic paradigm of Pregel.

- In Section 5, we report on experimental results with our Pregel-based algorithm. The empirical results confirm that parallelization via a vertex-centric approach indeed works very well in practice.

Related work. We briefly review related work on algorithms for preference aggregation and winner determination. Most work in this direction is focused on NP-hard voting rules, in particular the Kemeny rule (see, e.g., [Conitzer *et al.*, 2006; Betzler *et al.*, 2014; Schalekamp and van Zuylen, 2009; Ali and Meilă, 2012]).

For some voting rules the complexity changes depending on whether a fixed tie-breaking order is used. This is the case for the STV rule, where the winner determination problem is NP-hard if no tie-breaking order is specified [Conitzer *et al.*, 2009] (i.e., for the decision problem whether there is a tie-breaking order such that a distinguished candidate wins subject to this tie-breaking order), but STV is P-complete for a fixed tie-breaking order [Csar *et al.*, 2017]. Similarly, the ranked pairs method is NP-hard to compute without specified tie-breaking [Brill and Fischer, 2012]. We show in this paper that ranked pairs winner determination for a fixed tie-breaking order is P-complete. Recent work by Jiang *et al.* [2017] has considered the NP-hard variants of STV and ranked pairs and established fast algorithms for these problems. The use of parallel algorithms for winner determination has been previously studied by Csar *et al.* [2017] in the MapReduce framework, in particular for tournament solution concepts. Finally, we remark that Parkes and Xia [2012] considered similarities and differences of the ranked pairs and Schulze method in the context of strategic voting.

2 Preliminaries

A directed graph (digraph) is a pair (V, E) with $E \subseteq V \times V$. A path (from x_1 to x_k) is a sequence $\pi = (x_1, \ldots, x_k)$ of vertices with $(x_i, x_{i+1}) \in E$ for every $i \in \{1, \ldots, k-1\}$. We call π a cycle, if $x_1 = x_k$. A digraph without cycles is referred to as DAG (directed acyclic graph).

A set $X \subseteq V$ of vertices is called strongly connected if for every pair (a, b) of vertices in X, there is a path from a to b and from b to a. If X is maximal with this property, we call it a strongly connected component (SCC).

A weighted digraph is a triple (V, E, w) with (V, E) being a digraph and the function $w \colon E \to [0, \infty)$ assigning (non-negative) weights to edges.

2.1 Voting

Let A be a set of alternatives (or candidates) and $N = \{1, \ldots, n\}$ a set of voters. A preference profile $P = (\succeq_1, \ldots, \succeq_n)$ contains the preferences of the voters. We require $\succeq_1, \ldots, \succeq_n$ to be weak orders on A, i.e., transitive and complete relations. We write $\succ_i$ to denote the strict part of $\succeq_i$.

We now define several concepts based on a given preference profile P. We say that alternative a dominates alternative b if more voters prefer a to b than b to a, i.e., $|i \in N : a \succ_i b| > |i \in N : b \succ_i a|$; let $D_P \subseteq A \times A$ denote the corresponding dominance relation. The (strict) *dominance graph* is the digraph (A, D_P), i.e., there exists an edge from vertex a to b if and only if a dominates b. The *majority margin* of two candidates a, b is defined as $\mu_P(a, b) = |i \in N : a \succ_i b| - |i \in N : b \succ_i a|$. Let $\mathcal{W}_P = (A, E_P, \mu'_P)$ with $E_P = \{(a, b) \in A^2 : \mu_P(a, b) > 0\}$ and μ'_P being the restriction of μ_P to E_P. Note that $\mathcal{W}_P$ is a weighted digraph; we refer to it as the *weighted tournament graph of P*. We refer to elements of A interchangeably as candidates or vertices.

The *Schwartz set* is defined as the union of all non-dominated SCCs in the dominance graph. The *Schulze method* [2011] is a refinement of the Schwartz method. Its definition depends on *widest paths* in weighted graphs. Let (A, E, μ) be a weighted tournament graph. A path $(x_1, \ldots, x_k)$ has *width* α if $\min_{i \in \{1, \ldots, k-1\}} \mu(x_i, x_{i+1}) = \alpha$. A *widest path from a to b* is a path from a to b of maximum width; let $p(a, b)$ denote the width of such a path. According to the Schulze method, an alternative a beats alternative b if there is a wider path from a to b than from b to a, i.e., if $p(a, b) > p(b, a)$. An alternative a is a Schulze winner if there is no alternative b that beats a. The Schulze method can also be used to compute an output ranking, which is defined by the relation $(a, b) \in R$ if and only if $p(a, b) \geq p(b, a)$. It can be shown that R is a weak order [Schulze, 2011]. The Schulze winners are exactly the top-ranked alternatives in R.

We define the *ranked pairs method* [Tideman, 1987] subject to a fixed tie-breaking order[3] T, which is a linear order of the candidates. The ranked pairs method creates a ranking, starting with an empty relation R. All pairs of candidates are sorted according to their majority margin and ties are broken according to T. Then, pairs of candidates are added to the relation R in the sorted order (starting with the largest majority margin). However, a pair is omitted if it would create a cycle

---

[3]We note that the ranked pairs method satisfies the independence of clones property only for specific tie-breaking orders [Zavist and Tideman, 1989]; our results are in particular applicable to these distinguished tie-breaking orders.
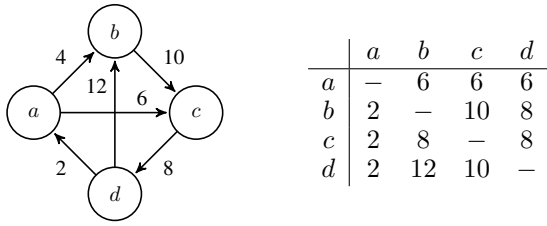
	a	b	c	d
a	–	6	6	6
b	2	–	10	8
c	2	8	–	8
d	2	12	10	–

Figure 1: A weighted tournament graph and its widest paths.

in R. The final relation R is a ranking of all alternatives; the top-ranked alternative is the winner (subject to T).

Example 1 Consider the weighted tournament graph displayed in Figure 1 on the left (originally by Schulze [2003]). The table in Figure 1 on the right shows the widest paths between any two vertices. The unique *Schulze winner* is candidate a, having a path of width 6 to every other candidate, whereas all incoming paths to vertex a have width 2.

The unique *ranked pairs winner* is d (independently of the chosen tie-breaking order). It is obtained by inspecting the edges in descending order of weights and retaining an edge only if it does not cause a cycle. We thus retain $(d, b), (b, c)$, omit (c, d), and retain $(a, c), (a, b), (d, a)$. Hence, d is the only vertex without incoming edge in the resulting DAG.

The digraph in Figure 1 is strongly connected. Hence, the set of *Schwartz winners* is the entire SCC $\{a, b, c, d\}$.   ⋄

2.2 Cloud Computing Frameworks

Cloud computing algorithms are based on the concept of splitting problem instances into small parts and performing computations on those parts using independent computation nodes. This approach gained popularity with the MapReduce framework [Dean and Ghemawat, 2008], which is ideal for distributed batch processing; for a general overview of MapReduce and its variants we refer the reader to a survey by Sakr et al. [2013]. The related Pregel framework [Malewicz *et al.*, 2010] was introduced specifically for big data problems on huge graphs.

Pregel algorithms are vertex-centric computations. They are often described as 'think-like-a-vertex' algorithms, where each vertex acts as an independent computation entity. This means that the vertices are distributed among the nodes of the cluster and the computations at each vertex can be performed in parallel. The vertices exchange information by sending messages to each other along the edges of the graph. Moreover, each vertex can store its own local information.

The Pregel computation works in supersteps; at the beginning of each superstep, the vertices read messages sent by other vertices in the previous superstep. If a vertex does not receive any messages, it is set inactive (but can be reactivated by messages in following supersteps); Pregel programs terminate if all vertices are inactive. After receiving messages, a vertex performs its vertex program, in which the information stored at the vertex can be changed and messages can be sent to other vertices. Finally, a message combiner can be used to collect and combine messages in order to optimise the communication between vertices.

3 Computational Complexity

In this section, we classify the computational complexity of the Schulze method and the ranked pairs method. In both cases, we do not need a preference profile as input but only the corresponding weighted majority graph. We thus define the winner determination problems for $\mathcal{R} \in \{\text{Schulze},$ Ranked Pairs$\}$ as follows:

$\mathcal{R}$ WINNER DETERMINATION	
Instance:	a weighted tournament graph $\mathcal{W} = (A, E, \mu)$, a candidate c
Question:	Is c a winner in $\mathcal{W}$ according to rule $\mathcal{R}$?

The main results of this section are the following:

Theorem 1 *The* SCHULZE WINNER DETERMINATION *problem is* NL-*complete.*

Theorem 2 *The* RANKED PAIRS WINNER DETERMINATION *problem is* P-*complete.*

P-membership in case of the ranked pairs method is obvious (and well-known). It remains to prove the remaining three properties: NL-membership, NL-hardness, and P-hardness.

In the following hardness proofs, we will construct weighted tournament graphs $\mathcal{W} = (A, E, \mu)$ with even integer weights and with E being an asymmetric relation, i.e, if $(a, b) \in E$ then $(b, a) \notin E$. It follows from McGarvey's Theorem [McGarvey, 1953] that such weighted tournament graphs can be obtained even from preference profiles containing only $\sum_{e \in E} \mu(e)$ many linear orders. Of course, our results also hold if preference profiles are given as input.

Our **NL-membership** proof for the Schulze-Winner Determination Problem is based on the following two related problems WP$_{\geq}$ and WP$_{>}$, where we ask if, for given vertices s, t and width w, a path from s to t of width $\geq w$ or $> w$, respectively, exists. Formally, we study the following problems:

EXISTENCE-OF-WIDE-PATHS WP$_{\geq}$ / WP$_{>}$	
Instance:	a weighted graph $\mathcal{G}$, vertices s, t, weight $w \in \mathbb{R}$
Question:	Does there exist a path from s to t of width $\geq w$ (in case of the WP$_{\geq}$-problem) or of width $> w$ (in case of the WP$_{>}$-problem)?

It is straightforward to verify that both problems WP$_{\geq}$ and WP$_{>}$ are in NL: guess one vertex after the other of a path from s to t and check for any two successive vertices that they are connected by an edge of weight $\geq w$ or $> w$, respectively.

Since NL = co-NL by the famous Immerman-Szelepcsényi Theorem, the co-problems of WP$_{\geq}$ and WP$_{>}$ are also in NL. We can thus construct a non-deterministic Turing machine (NTM) for the Schulze winner determination problem, which loops trough all candidates $c' \neq c$ and does the following:

- guess the width w of the widest path from c to c' (among the weights of the edges in $\mathcal{W}$);
- solve the WP$_{\geq}$ problem for vertices c, c' and weight w: check that there exists a path from c to c' of width $\geq w$;
- solve the co-problem of WP$_{>}$ for c', c and weight w: check that there is *no* path from c' to c of width $> w$;

The correctness of the NTM is immediate. Moreover, by the above considerations on the problems $\text{WP}_\geq$, $\text{WP}_>$ and their co-problems, the NTM clearly works in log-space time.

Note that if we want to check if c is the *unique* Schulze winner, then we just need to replace the third step above by a check that there is no path from c' to c of weight $\geq w$; in other words, we solve the co-problem of $\text{WP}_\geq$. Again, the overall NTM clearly works in log-space.

We prove **NL-hardness** by reduction from the NL-complete Reachability problem[4]. Let $(\mathcal{G}, a, b)$ be an arbitrary instance of Reachability with $\mathcal{G} = (V, E)$ and $a, b \in V$. We construct a weighted tournament graph $\mathcal{W} = (V', E', \mu)$ from $\mathcal{G}$ as follows and choose a as the distinguished candidate:

- First, remove from $\mathcal{G}$ all edges of the form (v, a) and (b, v) for every $v \in V$, i.e., all incoming edges of a and all outgoing edges from b.
- For every pair of symmetric edges $e_1 = (v_i, v_j)$ and $e_2 = (v_j, v_i)$, choose one of these edges (say e_1) and introduce a "midpoint", i.e., add vertex w_{ij} to V and replace e_1 by the two edges (v_i, w_{ij}) and (w_{ij}, v_j).
- For every vertex u different from b, introduce a new vertex r_u and edges (b, r_u), (r_u, u).
- Now there is exactly one incoming edge of a, namely $e = (r_a, a)$. Define the weight of this edge as $\mu(e) = 2$ and set $\mu(e') = 4$ for every other edge $e' \neq e$.

It is easy to verify that a is a Schulze winner in $\mathcal{W}$ (actually, it is even the unique Schulze winner), if and only if there is a path from a to b in $\mathcal{G}$. To see this, first observe that there is a path from a to b in $\mathcal{W}$ if and only if there is one in $\mathcal{G}$. Hence, if there is a path from a to b, then the widest path from a to any vertex in $\mathcal{W}$ is 4. Conversely, all paths from any vertex to a must go through edge (r_a, a) and, therefore, have width at most 2. On the other hand, if there is no path from a to b, then a cannot be a winner, since b indeed has a path to a (via r_a) and, therefore, in this case, b is definitely preferred to candidate a according to the Schulze method.

The **P-hardness** proof is by reduction from the classical P-complete Boolean-Formula-Evaluation (BFE) problem: given a Boolean formula ϕ with variables in X and truth assignment I on X, does $I \models \phi$ hold?

Consider an arbitrary instance (ϕ, X, I) of the BFE problem. W.l.o.g., we may assume that (1) each variable in ϕ occurs exactly once in ϕ and (2) the only connective in ϕ is nor. We construct an instance of the Ranked Pairs Winner Determination Problem $(\mathcal{W}, v)$ with $\mathcal{W} = (A, E, \mu)$ as follows:

Consider the parse tree $\mathcal{T}$ of ϕ. Each node of $\mathcal{T}$ corresponds to a subexpression of ϕ. In particular, the leaf nodes correspond to the variables in ϕ. Now let $\{g_1, \ldots, g_m\}$ denote the inner nodes of $\mathcal{T}$ in some *top-down* order, i.e., whenever g_i is an ancestor of g_j, then $i < j$ holds. Moreover,

[4]Brandt *et al.* [2009] show that any rule that selects winners from the Schwartz set—as Schulze's method does—and that break ties among candidates according to a fixed order is NL-hard to compute. Our result is similar but shows NL-hardness without requiring a tie-breaking order.

let $X = \{x_1, \ldots, x_\ell\}$ and let $\{g_{m+1}, \ldots, g_{m+\ell}\}$ denote the corresponding leaf nodes in $\mathcal{T}$. Then we set

$$A = \{g_0\} \cup \{g_1, \ldots, g_m, g_{m+1}, \ldots, g_{m+\ell}\} \cup \{h_1, \ldots, h_m\}.$$

By slight abuse of notation, we use $g_1, \ldots g_{m+\ell}$, to denote (1) candidates/vertices in $\mathcal{W}$, (2) nodes in $\mathcal{T}$, and (3) subexpressions of ϕ. The vertices $g_0, h_1, \ldots, h_m$ are new symbols. We say that "g_j is the parent of g_i", if g_j is the parent of g_i in the parse tree $\mathcal{T}$. In addition, we define g_0 as the "parent of g_1".

To define the set of edges E together with weight function μ, let $\alpha \in \{1, \ldots, m + \ell\}$; we distinguish two cases:

Case 1: Suppose $\alpha \in \{1, \ldots, m\}$, i.e., g_α is an inner node of $\mathcal{T}$. Let g_i be the parent of g_α and, for $\alpha \neq 1$, let g_j be the parent of g_i. Then E contains the following edges:
$e_0 = (g_j, g_\alpha)$ with $\mu(e_0) = 4\alpha$,   for every $\alpha \neq 1$.
$e_1 = (g_\alpha, h_\alpha)$ with $\mu(e_1) = 4\alpha - 1$.
$e_2 = (h_\alpha, g_i)$ with $\mu(e_2) = 4\alpha - 2$.
$e_3 = (g_i, g_\alpha)$ with $\mu(e_3) = 4\alpha - 3$.

Case 2: Suppose $\alpha \in \{m + 1, \ldots, m + \ell\}$, i.e., g_α is a leaf node in $\mathcal{T}$ (corresponding to some variable x_γ). Again, let g_i denote the parent of g_α and let g_j denote the parent of g_i. Then E contains $e_0 = (g_j, g_\alpha)$ with $\mu(e_0) = 4m + 2\gamma$. Moreover, if $I(x_\gamma) = \text{true}$, then E contains $e_1 = (g_\alpha, g_i)$. Conversely, if $I(x_\gamma) = \text{false}$, then E contains $e_1 = (g_i, g_\alpha)$. In either case, we set $\mu(e_1) = 4m + 2\gamma - 1$.

These are all the edges in E. For the sake of readability, we use the integer interval from 1 to $4m + 2\ell$ as weights (with weight 4 missing, since g_1 has no grand-parent). The weights could be easily made even by multiplying all weights by 2. Finally, we choose g_1 as the distinguished vertex for which we have to decide whether it is a ranked pairs winner.

The intuition of this construction is as follows: in the ranked pairs method, we inspect the edges in descending order of their weights (note that all edges in $\mathcal{W}$ have unique weights) and check if the current edge may be added to the relation R (now considered as a DAG) without producing a cycle. Then R contains either the edge (g_j, g_i) or a path from g_i to g_j (in case g_i is an inner node of $\mathcal{T}$, this is the path $g_i \rightarrow h_i \rightarrow g_j$; in case g_i is a leaf node, this is the edge (g_i, g_j)). The crucial property of our construction (which can be proved by structural induction on the nodes of the parse tree $\mathcal{T}$) is that the existence of a path from g_i to g_j encodes $I \models g_i$ (recall that we identify nodes in the parse tree with the corresponding subexpressions of ϕ), whereas the existence of the edge (g_j, g_i) in R encodes $I \not\models g_i$.

In particular, for g_1 with parent g_0 this means that if $I \not\models g_1$, then R contains the edge (g_0, g_1) and g_1 is clearly not a winner. Conversely, suppose that $I \models g_1$ holds and let $g_1 = g_\alpha \text{ nor } g_\beta$. Of course, $I \models g_1$ means that $I \not\models g_\alpha$ and $I \not\models g_\beta$. Hence, R contains no incoming edge to g_1 at all. Hence, in this case, g_1 is a winner (actually even the unique winner) independently of the chosen tie-breaking order.

The construction is illustrated by Example 2.

Example 2 *Consider formula* $\phi = x_1 \text{ nor } (x_2 \text{ nor } x_3)$ *and assignment I with $I(x_1) = I(x_2) = \text{true}$ and $I(x_3) = \text{false}$. The parse tree $\mathcal{T}$ of ϕ has 2 inner nodes (g_1, g_2) corresponding to the two occurrences of* nor *and 3 leaf nodes (g_3, g_4, g_5)*
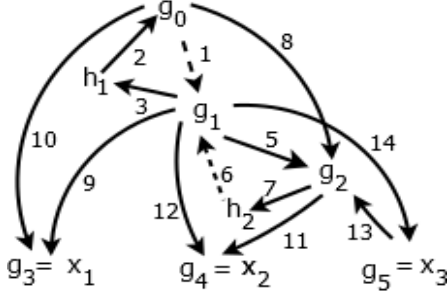
Figure 2: Graph $\mathcal{W}$ and relation R of Example 2 for Boolean formula $\phi = x_1$ nor $(x_2$ nor $x_3)$ and assignment I with $I(x_1) = I(x_2) =$ true and $I(x_3) =$ false.

for the 3 variables. The graph $\mathcal{W}$ resulting from our problem reduction and the DAG R resulting from applying the ranked pairs method are displayed in Figure 2, where $\mathcal{W}$ contains all edges shown in the figure while R contains only the edges with solid lines.

The edges to the leaf nodes x_1, x_2 and from leaf node x_3 directly encode the truth value of these variables in I. For each inner node g_α (for $\alpha \in \{1, 2\}$) with parent g_i, the graph $\mathcal{W}$ contains the triangle g_α, h_α, g_i. Depending on the truth value of g_α, either edge (h_α, g_i) is retained in R (if $I \models g_\alpha$) or edge (g_i, g_α) (if $I \not\models g_\alpha$). Formula ϕ evaluates to true for the assignment I and g_1 is indeed the unique winner in the election represented by $\mathcal{W}$.

4 Computation of the Schulze Winner

In this section, we present our new Pregel-based algorithm for determining the Schulze winners for a given weighted tournament graph $\mathcal{W} = (A, E, \mu)$. A straightforward algorithm, which is implicit in the NL-membership proof in Section 3, would consist in computing for every pair (a, b) of vertices in $\mathcal{W}$ the widths $p(a, b)$ and $p(b, a)$. However, this would mean that we have to store for each vertex c a linear amount of information (namely $p(c, v)$ for every $v \in A$). This contradicts the philosophy of Pregel algorithms which aim at keeping the local information at each vertex small [Yan *et al.*, 2014].

An improvement of this idea would be to first compute the SCCs of $\mathcal{W}$ by one of the Pregel algorithms in the literature [Salihoglu and Widom, 2014; Yan *et al.*, 2014] and to compute the widest paths for pairs of vertices only for the non-dominated SCCs (i.e., the Schwartz winners). This approach has the disadvantage that the set of pairs to be considered may still be big and, moreover, the computation of the SCCs only makes use of part of the information (namely the dominance graph—thus ignoring the weights of the edges).

We therefore construct a new Pregel-style algorithm, which draws some inspiration from the well-studied SCC computation (in particular, the forward/backward propagation of minimum vertex-ids) but utilises the weight information to prune the search space as soon as possible.

The proposed Pregel algorithm for computing the Schulze Method is guaranteed to have very small local information at each vertex. The overall structure of our algorithm is given in Algorithm 1.

Algorithm 1 Schulze Winner Determination

Initialisation-of-vertices;
while there exists a vertex c with $c.status = $ 'unknown' **do**
    Preprocessing;
    Forward-Backward-Propagation;
    Postprocessing;
Output vertices with status 'winner';

We assume that each vertex c is assigned a unique id $c.id \in \{1, \dots m\}$. Moreover, each vertex c has a *status* which may take one of 3 values {'winner', 'loser', 'unknown'} to express that c is a Schulze winner, not a Schulze winner, or if we do not know yet, respectively. Additional information stored at each vertex includes the fields s, t, ws, wt, and scc, whose meaning will be explained below, as well as information on the adjacent edges together with their weights.

In **Initialisation-of-vertices**, we determine for each candidate c the maximum weight of all incoming and outgoing edges and set the status accordingly: if $\max_{a \in A} \mu(a, c) > \max_{a \in A} \mu(c, a)$, then there exists a vertex v (namely the one with $\mu(v, c) = \max_{a \in A} \mu(a, c)$) which is preferred to c by the Schulze method. Hence, in this case, we set $c.status = $ 'loser'; otherwise we set $c.status = $ 'unknown';

The goal of each iteration of the **while-loop** in Algorithm 1 is to compute for every vertex c the ids s (= source) and t (= target) which are the minimum ids among all vertices with $status = $ 'unknown' such that there is a path from s to c and from c to t. Moreover, we also determine the weights ws and wt of the widest paths from s to c and from c to t. Termination is guaranteed since in each iteration at least one vertex changes its status from 'unknown' to 'loser' or 'winner'. Our experimental evaluation shows that the algorithm terminates very fast: on real-world data, typically even a single iteration of the while-loop suffices. Preliminary experiments with synthetic data show that the while-loop is executed less than 10 times for instances with 10.000 candidates.

In **Preprocessing**, shown in Algorithm 2, we initialise the fields (s, ws, t, wt) of all vertices. For every vertex c with $c.status = $ 'unknown', we set $c.s = c.t = c.id$. Thus, initially, the minimum id of vertices to reach c and reachable from c is the id of c itself. We send the information on c as a source (resp. target) to its adjacent vertices via outgoing (resp. incoming) edges. For vertices with status different from 'unknown', we set $c.s = c.t = \infty$. This allows such a vertex c to pass on vertex ids from other sources and targets but it prevents c from passing on its own id.

Algorithm 2 Preprocessing

---

if $c.status = $ 'unknown' **then**
    $s = c.id; ws = \infty; t = c.id; wt = \infty;$
    for each outgoing edge (c, v) with weight w **do**
        send ('forward', s, w) to vertex v;
    for each incoming edge (v, c) with weight w **do**
        send ('backward', t, w) to vertex v;
else
    $s = \infty; ws = 0; t = \infty; wt = 0;$

Algorithm 3 Forward-Backward-Propagation

> **for each** received value (d, v, w) **do**;
>   **if** $d = $ 'forward' **then**
>   **if** $v < s$ **then** $s = v$; $ws = w$;
>   **else if** $v = s$ **then** $ws = \max(ws, w)$;
>   **else if** $d = $ 'backward' **then**
>   **if** $v < t$ **then** $t = v$; $wt = w$;
>   **else if** $v = t$ **then** $wt = \max(wt, w)$;
> **if** (s, ws) has changed **then**
>   **for each** outgoing edge (c, v) with weight w **do**
>   send ('forward', s, $\min(ws, w)$) to vertex v;
> **if** (t, wt) has changed **then**
>   **for each** incoming edge (v, c) with weight w **do**
>   send ('backward', t, $\min(wt, w)$) to vertex v;
> set c inactive;

Algorithm 4 Postprocessing for vertex c

> **if** $c.s < c.t$ **then**
> $c.status = $ 'loser';
> **else if** $c.s > c.t$ **then**
>   set $status$ of vertex $c.t$ to 'loser';
> **else if** $c.s = c.t$ **then**
> $c.scc = c.s$;
>   **if** $c.ws > c.wt$ **then** $c.status = $ 'loser';
>   **else if** $c.ws < c.wt$ **then** set $status$ of vertex $c.s$ to 'loser';
> **for each** incoming edge (v, c) **do**
>   get $(v.s, v.t)$ from vertex v;
>   **if** $c.s \neq v.s$ **or** $c.t \neq v.t$ **then**
>   **if** $c.s = c.t$ **then**
>   **for each** vertex u with $u.scc = c.s$ **do**
>   set $status$ of vertex u to 'loser';
>   **else if** $c.s \neq c.t$ **then**
> $c.status = $ 'loser';
> **if** $c.scc = c$ **and** $c.status = $ 'unknown' **then**
> $c.status = $ 'winner';

The **Forward-Backward-Propagation** is the actual 'Pregel heart' of the computation. The other procedures work in parallel too, but they do not use the Pregel Computation API. Algorithm 3 realizes the forward and backward propagation as a Pregel procedure. For each vertex c, we determine (1) the minimum source-id s together with the maximum width ws of paths from s to c and (2) the minimum target-id t together with the maximum width wt of paths from c to t. We thus analyse each received message (d, v, w) consisting of a direction d, vertex id v, and width w. In case of 'forward' direction, we have to check if we have found a source v with a yet smaller id than the current value s. If so, we update $c.s$ and $c.ws$ accordingly. If the received vertex-id v is equal to the current value of $c.s$, we have to update $c.ws$ in case the received value w is greater than $c.ws$ (i.e., from the same source we have found a path of greater width).

Messages in 'backward' direction are processed analogously, resulting in possible updates of the target-id t and/or the width wt of paths from c to t. After all messages have been processed, we propagate the information on new source id s (resp. target id t) and/or increased width of paths from s to c (resp. from c to t) to all adjacent vertices of c in forward (resp. backward) direction. Forward-Backward-Propagation terminates when no more messages are pending.

In **Postprocessing**, as shown in Algorithm 4, we use two crucial properties of source and target ids, which are inherited from the SCC computation in [Yan *et al.*, 2014]: First, if for a vertex c, we have $c.s = c.t$, then the set of vertices v with the same source/target id (i.e., $v.s = v.t = c.s$) forms the SCC of c. Second, if for two vertices c and v, we have $c.s \neq v.s$ or $c.t \neq v.t$, then c and v belong to two different SCCs.

In Algorithm 4, we first compare for each vertex c the values of s and t: if $c.s < c.t$, then c is reachable from s but s is not reachable from c. Hence, c is a loser. If $c.s > c.t$, then t is reachable from c but c is not reachable from t. Hence, t is a loser. Note that setting the status of t (which is, in general, different from the current node) is done by a subroutine whose details are omitted here. Finally, if $c.s = c.t$, then (as recalled above) we have found the SCC of c. As in [Yan *et al.*, 2014], we use the minimum id of the vertices in an SCC to label the SCC. If the width of the path from s (which is

equal to t by our case distinction) to c is greater than from c to t, then c is a loser (since s is preferred to it). In the opposite case, s is a loser.

In the next step in Algorithm 4, we compare the values of (s, t) of each vertex c with the values of $(v.s, v.t)$ of all vertices v with an edge (v, c). If $v.s \neq c.s$ or $v.t \neq c.t$, then v and c are in different SCCs. By the existence of the edge (v, c), this means that there can be no path from c to v. Hence, v is preferred to c according to the Schulze method. Moreover, if $c.s = c.t$, then we have found the SCC of c. In this case, v is preferred to all vertices u in this SCC.

Finally, suppose that we have found some SCC such that the vertex c with minimum id in this SCC has not been identified as a loser by any of the above cases. In particular, this means that none of the vertices in this SCC has an incoming edge from outside the SCC and, moreover, the SCC cannot contain a vertex v with $p(v, c) > p(c, v)$. In this case, we may mark vertex c as a winner. It is now also clear that at least one vertex must change its status from 'unknown' to either 'loser' or 'winner' in every execution of Algorithm 4 and, therefore, in every iteration of the while-loop of Algorithm 1.

We briefly describe further **optimisations**. For instance, rather than assigning vertex-ids randomly, we first perform the cheap computation of Borda scores and assign vertex-ids in descending order of Borda scores. This yields lower ids for vertices that are more likely to dominate other vertices, thus speeding up the exclusion of dominated candidates. Moreover, in the postprocessing phase, we exclude all vertices of a dominated SCC from further consideration simply by setting the weights of all incoming edges to this SCC to 0. This is much faster than physically deleting these vertices, since in GraphX, changing the graph topology is time intensive.

Finally, we remark that by iterating the algorithm k times and continuously removing Schulze winners, it is straightforward to compute a top-k ranking according to Schulze.

Table 1: Spotify data sets

	candidates	voters	edges	after preproc.
Europe150	9698	7,481	44.1M	11 undecided
Europe200	12250	7,481	70.7M	12 undecided
Global150	14187	15,553	94.9M	8 undecided
Global200	18407	15,553	159.6M	9 undecided

5 Experiments

5.1 Setup and Data

To assess the performance of big data algorithms, it is essential to test them against actual real-world large-scale data sets. To this end, we use the Spotify ranking data[5] of 2017, which consists of daily top-200 music rankings for 53 countries. We consider the ranking of each day and country as a single voter, and then generate the corresponding weighted tournament graph. Our experiments are based on four data sets generated from this data: Global150, Global200, Europe150, and Europe200, which are based on daily top-150/top-200 charts of all available/European countries. We do not take into account the number of listeners in each country, since this information is not available—but this could easily be included in our computation by assigning countries weights that correspond to the number of listeners. In Table 1, we provide an overview of these four data sets. All four weighted tournament graphs are dense: edges exist between roughly 94% of all candidate pairs. We note that the Spotify data sets used here are significantly larger than any instances available in the PrefLib database [Mattei and Walsh, 2013] and the data sets used by Csar et al. [2017], which have $\leq$ 7000 vertices and less than 5 million edges.

We ran our experiments on a Hadoop cluster with 18 nodes (each with an Intel Gold 5118 CPU, 12 cores, 2,3 GHz processor, 256 GB RAM, and a 10Gb/s network connection). To better observe the scalability of our algorithm, we restricted the number of cores and nodes (details follow).

Our Schulze algorithm is implemented in the Scala programming language. Furthermore, we use the GraphX library[6], which is built on top of Spark [Zaharia *et al.*, 2010], an open-source cluster-computing engine. GraphX provides a Pregel API, but is slightly more restrictive than the Pregel framework. In particular, in GraphX, only messages to adjacent vertices can be sent, while other Pregel implementations allow messages to be sent to arbitrary vertices. The source code of our implementation is part of the open-source project CloudVoting[7].

5.2 Results

Our experiments show that our algorithm scales very well with additional computational resources: both an increase in nodes and in cores per node significantly sped up the computation. We refer the reader to Figure 3 for an overview of runtimes for 1/2/3/4 nodes with 1/2/4/8 cores each. On

---

[5]https://spotifycharts.com/regional
[6]https://spark.apache.org/graphx/
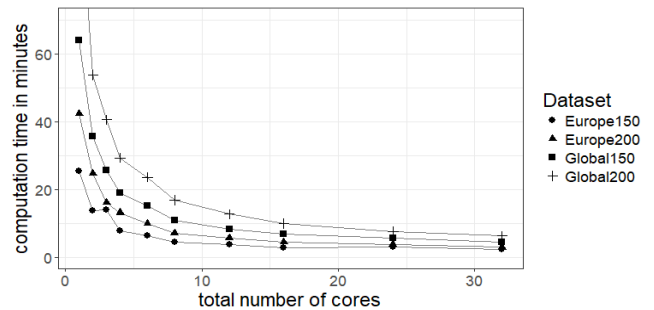[7]https://github.com/theresacsar/CloudVoting

Figure 3: Runtime required for computing Schulze winners.

the x-axis of this chart we show the total number of cores, i.e., the number of nodes times the number of cores per node. For x-values with multiple interpretations we show the best runtime. This is always the configuration with most cores per node, but the differences between an increase in nodes or cores is almost negligible. Furthermore, our implementation manages to compute Schulze winners of all Spotify data sets within very reasonable time: with 4 nodes each using 8 cores, the data sets could be handled in less than 6.5min.

It is insightful to compare the runtimes of our algorithm with the original, sequential algorithm [Schulze, 2011] based on the classical Floyd–Warshall algorithm; to the best of our knowledge this is the only published algorithm computing the Schulze method. These two algorithms differ not only in their capability of parallelization, but also in that our algorithm only returns Schulze winners whereas the original algorithm returns a full ranking of candidates. Due to our focus on Schulze winners, we could include the many optimisations described in Section 4. As a consequence of these optimisations and our focus on winners, our algorithm is faster than the original algorithm even without parallelization (1 node with 1 core): Our algorithm requires with this configuration 26min/42min/64min/105min for the the Europe150/200 and Global150/200 data sets, respectively. In contrast, the original algorithm (also implemented in Scala) requires 71min/143min/221min for the Europe150/200 and Global150/200 data sets; it did not terminate in reasonable time for the Global200 data set and we stopped the computation after 5 hours. As mentioned before, this comparison is not completely fair due to the different output, but shows the impact of the optimisations in our algorithm.

We also performed preliminary experiments with low-density graphs (based on synthetic data). We observed that for these graphs the number of undecided candidates decreases slower with each iteration of the forward-backward propagation, in contrast to the Spotify data sets where even after preprocessing very few undecided candidates remained (cf. Table 1). However, for 10,000 candidates we obtain comparable runtimes to the Spotify data sets and hence are optimistic that our algorithm behaves rather robustly with respect to varying densities.

6 Conclusion and Directions for Future Work

This paper shows the great potential of parallel algorithms and cloud computing techniques in computational social

choice, but many challenges remain. During the experimental evaluation we experienced that it is a non-trivial task to generate synthetic data sets sufficiently large for benchmark purposes. In particular, it is challenging to generate preference profiles according to the widely used Mallows model with more than 10,000 candidates. The compilation of large real-world data sets is equally important, as most preference data sets currently available are insufficient for large-scale experiments. As further future work we plan to investigate other voting rules with respect to their parallelizability and their suitability for handling large preference data sets.

Acknowledgments

References

[Ali and Meilă, 2012] Alnur Ali and Marina Meilă. Experiments with kemeny ranking: What works when? *Mathematical Social Sciences*, 64(1):28–40, 2012.

[Betzler *et al.*, 2014] Nadja Betzler, Robert Bredereck, and Rolf Niedermeier. Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation. *Autonomous Agents and Multi-Agent Systems*, 28(5):721–748, 2014.

[Brandt *et al.*, 2009] Felix Brandt, Felix Fischer, and Paul Harrenstein. The computational complexity of choice sets. *Mathematical Logic Quarterly*, 55(4):444–459, 2009.

[Brandt *et al.*, 2016] Felix Brandt, Markus Brill, and Paul Harrenstein. Tournament solutions. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel Procaccia, editors, *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[Brill and Fischer, 2012] Markus Brill and Felix Fischer. The price of neutrality for the ranked pairs method. In *Proceedings of AAAI-12*, 2012.

[Conitzer *et al.*, 2006] Vincent Conitzer, Andrew Davenport, and Jayant Kalagnanam. Improved bounds for computing Kemeny rankings. In *Proceedings of AAAI-06*, pages 620–626, 2006.

[Conitzer *et al.*, 2009] Vincent Conitzer, Matthew Rognlie, and Lirong Xia. Preference functions that score rankings and maximum likelihood estimation. In *Proceedings of IJCAI-09*, pages 109–115, 2009.

[Csar *et al.*, 2017] Theresa Csar, Martin Lackner, Reinhard Pichler, and Emanuel Sallinger. Winner determination in huge elections with MapReduce. In *Proceedings of AAAI-17*, 2017.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Jiang *et al.*, 2017] Chunheng Jiang, Sujoy Sikdar, Jun Wang, Lirong Xia, and Zhibing Zhao. Practical algorithms for computing stv and other multi-round voting rules. In *Proceedings of EXPLORE-17*, 2017.

[Malewicz *et al.*, 2010] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of SIGMOD-10*, pages 135–146. ACM, 2010.

[Mattei and Walsh, 2013] Nicholas Mattei and Toby Walsh. PrefLib: A library for preferences http://www.preflib.org. In *Proceedings of ADT-13*, pages 259–270, 2013.

[McGarvey, 1953] David C McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, 21(4):608–610, 1953.

[Parkes and Xia, 2012] David C Parkes and Lirong Xia. A complexity-of-strategic-behavior comparison between schulze's rule and ranked pairs. In *Proceedings of AAAI-12*, 2012.

[Sakr *et al.*, 2013] Sherif Sakr, Anna Liu, and Ayman G Fayoumi. The family of mapreduce and large-scale data processing systems. *ACM Computing Surveys*, 46(1):11, 2013.

[Salihoglu and Widom, 2014] Semih Salihoglu and Jennifer Widom. Optimizing graph algorithms on pregel-like systems. *Proceedings of VLDB-14*, 7(7):577–588, 2014.

[Schalekamp and van Zuylen, 2009] Frans Schalekamp and Anke van Zuylen. Rank aggregation: Together we're strong. In *Proceedings of ALENEX-2009*, pages 38–51, 2009.

[Schulze, 2003] Markus Schulze. A new monotonic and clone-independent single-winner election method. *Voting matters*, 17(1):9–19, 2003.

[Schulze, 2011] Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011.

[Tideman, 1987] T Nicolaus Tideman. Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.

[Yan *et al.*, 2014] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel algorithms for graph connectivity problems with performance guarantees. *Proceedings of VLDB-14*, 7(14):1821–1832, 2014.

[Zaharia *et al.*, 2010] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[Zavist and Tideman, 1989] Thomas M Zavist and T Nicolaus Tideman. Complete independence of clones in the ranked pairs rule. *Social Choice and Welfare*, 6(2):167–173, 1989.