Incomplete Preferences in Single-Peaked Electorates[*]

Martin Lackner
Vienna University of Technology, Austria
lackner@dbai.tuwien.ac.at

Abstract

Incomplete preferences are likely to arise in real-world preference aggregation and voting systems. This paper deals with determining whether an incomplete preference profile is single-peaked. This is essential information since many intractable voting problems become tractable for single-peaked profiles. We prove that for incomplete profiles the problem of determining single-peakedness is NP-complete. Despite this computational hardness result, we find two polynomial-time algorithms for reasonably restricted settings.

1 Introduction

Both human and automated decision making often have to rely on incomplete information. The same issue arises in joint decision making – voting – in multi-agent systems. Konczak and Lang [2005] distinguish two main sources of incompleteness: The first one is *intrinsic* incompleteness where the voter is unable or unwilling to give complete information, i.e., a total order on all candidates. The second one is *epistemic* incompleteness where the voters do have preferences specified by total orders but at the time of decision making these total orders are not fully available. Of course, a combination of these two scenarios is also possible.

Whereas complete preferences are usually modeled as total orders, incomplete preferences can be modeled as partial orders and are therefore a more general concept. In particular, the determination of winners becomes harder since voting protocols usually require total orders. It is therefore necessary to consider *completions* of incomplete votes which are total orders that are compatible extensions of the original partial orders. The determination of possible and necessary winners in incomplete elections is often NP-hard and thus a fast winner determination is not feasible [Konczak and Lang, 2005; Walsh, 2007; Betzler and Dorn, 2010; Pini *et al.*, 2011; Xia and Conitzer, 2011; Baumeister and Rothe, 2012].

A popular approach to deal with hardness of voting problems is to consider domain restrictions. The most common restriction is *single-peakedness* [Black, 1948] (see Section 2 for a definition). For example, computing the winner of a Dodgson or Kemeny election, though Θ_2^P-complete in general [Hemaspaandra *et al.*, 1997; 2005], can be done in polynomial time for single-peaked elections [Brandt *et al.*, 2010]. Also the complexity of manipulation and control problems often decreases [Faliszewski *et al.*, 2011b]. These results let us hope that efficient, polynomial time algorithms for computing possible and necessary winners of single-peaked, incomplete elections could be found. Walsh [2007] started investigating this issue and also pointed out a central question in that regard: What happens if the axis for which the incomplete preference profile is single-peaked is not given as part of the input but has to be determined?

Our paper deals with this question, namely how to determine single-peakedness for incomplete elections. In the following, let n denote the number of votes and let m denote the number of candidates. The main results are as follows:

• We prove that determining whether an incomplete preference profile is single-peaked is NP-complete. This is in contrast to the case of complete preferences for which single-peakedness can be determined in linear time [Escoffier *et al.*, 2008]. Furthermore, we strengthen this result by showing that NP-completeness still holds if a voter completely specifies his preferences. (Section 4)

Apart from these hardness results, this paper contains two polynomial time algorithms:

• (Guided Algorithm) The first algorithm requires that the preference profile possesses a so-called *guiding order*, an implicitly given completely specified vote. This is in particular the case for profiles actually containing a complete vote. The algorithm is applicable to weak orders (see Figure 1 for an example and Section 2 for a definition). We obtain a runtime of $\mathcal{O}(m \cdot n)$. This algorithm is an improvement over the algorithm by Escoffier *et al.* [2008] since it is applicable to a broader class of preference profiles (weak orders instead of total orders) while maintaining its runtime. (Section 5)

• (Unguided Algorithm) The second algorithm does not require a guiding order and is applicable to top orders. Top orders rank an arbitrary number of top candidates; all remaining candidates are ranked last and incomparable to one another (see Figure 1 for an example). This algorithm has a runtime of $\mathcal{O}(m^2 \cdot n)$. (Section 6)

• We have implemented the algorithms and performed benchmarks on randomly generated preference profiles. Even

[*]This work was supported by the Austrian Science Fund FWF, P25518-N23.

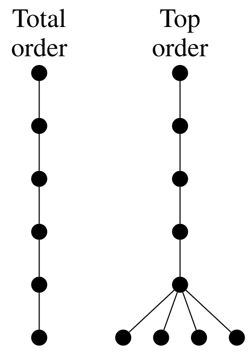

Figure 1: Examples of different types of orders that are used to specify preferences

large instances with 100,000 votes and 1,000 candidates can be handled with ease. (Section 7)

2 Preliminaries

In this paper, preferences are represented by different types of orders (see Figure 1 for examples). The most general type are partial orders. A *partial order* P on a set X is a reflexive, antisymmetric and transitive binary relation on X. We say that y *is ranked above* x if xPy holds. If for two elements $x, y \in X$ neither xPy nor yPx holds, these two elements are *incomparable*. A partial order where the incomparability relation is transitive is called a *weak order*. A weak order can thus be considered a total order with ties. Weak orders are also referred to as bucket orders (elements that tie are in the same "bucket"), e.g., in [Fagin *et al.*, 2006]. A weak order where every incomparable element is minimal is called *top order*. The *ranked* candidates of a top order T are those that are not incomparable to another candidate. We would like to remark that top orders appear as *top lists* in [Dwork *et al.*, 2001; Fagin *et al.*, 2003] and as top-truncated votes in [Baumeister *et al.*, 2012]. Finally, a partial order with no incomparable elements is called *total order*.

We use the notation $\langle c_1 > c_2 > \ldots > c_k \rangle$ to denote a top order where $c_1, \ldots, c_k$ are ranked as stated and all other elements are ranked last, i.e., are minimal elements. We use $\langle\rangle$ to denote the empty order relation, i.e., all elements are incomparable. We sometimes use set operators ($\cup, \cap, \setminus$) on top orders with the intended meaning that we apply these operators to the corresponding sets of ranked candidates.

We would now like to address the usefulness of these orders for expressing preferences. Total orders allow to fully specify a ranking of options. Given a large set of options, this might be unfeasible. Partial orders, on the other hand, allow to specify the relative order of any pair of options. Thus they can be seen as a very general formalism for representing incomplete preferences. They are compatible with total orders in the sense that partial orders can always be extended to total orders. Weak orders are less general than partial orders but arise in many natural scenarios. For example every real-valued utility function implies a weak order (candidates with the same utility tie, i.e., are incomparable). If the elicitation of preferences is costly, one might ask only for the most important (highest ranked) options of each voter. In such a case, top orders arise. Top orders also are the natural type of order for specifying preferences in some scoring protocols. We will further comment on scoring protocols and top orders in Section 8.

Throughout this paper we use C to denote the set of candidates or options. Votes are considered to be either partial, weak, top or total orders. For a vote V_i, we use $x \succ_i y$ to denote that $(yV_ix) \wedge (x \neq y)$, i.e., x is ranked strictly higher than y. If there is only one vote under consideration, usually denoted by V, we omit the index and write $x \succ y$. A tuple $(V_1, \ldots, V_n)$ of votes is called a *(preference) profile of* $\{$*partial orders, weak orders, top orders, total orders*$\}$, depending on the type of orders. Given a vote V and a set of candidates $C' \subseteq C$, we define $V[C']$ to be the vote V restricted to candidates in C'. Analogously, given a preference profile $\mathcal{P} = (V_1, \ldots, V_n)$, we define $\mathcal{P}[C']$ to be $(V_1[C'], \ldots, V_n[C'])$. We denote the number of candidates with m and the number of votes with n.

3 Single-peaked profiles

We start by giving a definition for single-peaked profiles of total orders and then extend this definition to partial orders. A central concept is that of an axis: this is a total order on C. Let A be an axis. Throughout this paper we write $x < y$ instead of xAy. (Note that we use $\succ$ for votes and $<$ for axes.) Our definition of single-peakedness for profiles of total orders as well as for profiles of partial orders is based on *valleys*.

Definition 1 (v-valleys)**.** Let V be a partial order on C. The vote V *contains a v-valley with respect to* A if there exist $c_1, c_2, c_3 \in C$ such that $c_1 < c_2 < c_3$, $c_1 \succ c_2$ and $c_3 \succ c_2$.

The definition of v-valleys suffices to define single-peakedness for profiles of total orders: A profile $\mathcal{P}$ of total orders is single-peaked with respect to A if no vote $V \in \mathcal{P}$ contains a v-valley with respect to A (and thus every vote has only a single "peak"). A profile of total orders is single-peaked consistent if there exists some axis A such that $\mathcal{P}$ is single-peaked with respect to A.

We now want to extend this definition to profiles of partial orders. The natural way is to consider extend the partial orders to total orders:

Definition 2. Let $\mathcal{P} = (V_1, \ldots, V_n)$ be a profile of partial orders. The profile $\mathcal{P}$ is single-peaked with respect to an axis A if for every $k \in \{1, \ldots, n\}$, V_k can be extended to a total order V'_k such that the profile of total orders $\mathcal{P}' = (V'_1, \ldots, V'_n)$ is single-peaked with respect to A.

This definition is impractical since it is hard to check. A definition based on valleys would be more preferable. This is possible but requires the definition of u-valleys in addition to v-valleys.

Definition 3 (u-valleys)**.** Let V be a partial order on C. The vote V *contains a u-valley with respect to* A if there exist distinct $a, b, c, d \in C$ with $a < b < d$ and $a \succ b$ as well as $a < c < d$ and $d \succ c$.

In Figure 2 a graphical representation of v- and u-valleys is shown. These two types of valleys allow a characterization of single-peakedness for profiles of partial orders.

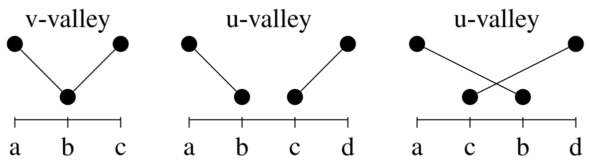

Figure 2: What v-valleys and u-valleys may look like.

Lemma 4. *Let $\mathcal{P} = (V_1, \ldots, V_n)$ be a profile of partial orders. The following two statements are equivalent.*

(i) The profile $\mathcal{P}$ is single-peaked with respect to A.

(ii) No vote $V \in \mathcal{P}$ contains either a u-valley or v-valley with respect to A.

Note that u-valleys cannot arise in weak orders. Thus, for profiles of weak orders, checking for the absence of v-valleys already suffices to verify single-peakedness.

Let $\mathcal{T} \in \{$partial order, weak order, top order, total order$\}$ be a type of order. In this paper we are going to study the $\mathcal{T}$ SINGLE-PEAKED CONSISTENCY problem, defined as follows: Given $\mathcal{P}$, a profile of type $\mathcal{T}$, and a set of candidates C, is $\mathcal{P}$ single-peaked consistent? It is well known that TOTAL ORDER SINGLE-PEAKED CONSISTENCY can be solved in polynomial time [Bartholdi and Trick, 1986; Doignon and Falmagne, 1994; Escoffier *et al.*, 2008]. In the next section, we show that this is likely not to be the case for partial orders.

4 Hardness results

Theorem 5. *The* PARTIAL ORDER SINGLE-PEAKED CONSISTENCY *problem is* NP*-complete.*

Proof. We reduce from the NP-complete BETWEENNESS problem [Opatrny, 1979]. A BETWEENNESS instance consists of a set S and a set T containing triples of distinct elements of S. The decision problem asks whether there is a total order L such that for every triple $(a, b, c) \in T$ we have either $aLbLc$ or $cLbLa$. Intuitively, a triple $(a, b, c) \in T$ corresponds to the constraint that b has to lie "in between" a and c on the total order L.

We construct an incomplete election $(C, \mathcal{P})$ with $C = S$, i.e., we identify elements in S with candidates. The preference profile $\mathcal{P}$ consists of two votes for each triple (a, b, c): the partial orders $\{a \succ c, b \succ c\}$ and $\{b \succ a, c \succ a\}$. These two votes form a valley on any axis with c between a and b and on any axis with a between b and c. Thus b has to be between a and c on any single-peaked axis. □

The proof of Theorem 5 uses elections where the votes contain very little information: only two pairs of candidates are comparable in each vote. We know that determining single-peaked consistency is possible in polynomial time if every vote is a total order, i.e., all votes contain complete information. Now the question arises: what happens if only a single voter provides complete information. Having a single completely specified voter has been found to be helpful in a related context: it allows to efficiently elicit single-peaked preferences using only few comparison queries [Conitzer, 2009] and thus reduces the communication complexity of preference elicitation. However, in our case such a voter does not provide enough additional information for decrease in (computational) complexity.

Theorem 6. *The* PARTIAL ORDER SINGLE-PEAKED CONSISTENCY *problem is* NP*-complete even if the preference profile contains a total order.*

Proof. We reduce from SET SPLITTING: Let X be a finite set. Given a collection C of subsets of X, is there a partition of X into two subsets X_1 and X_2 such that no subset of C is contained entirely in either X_1 or X_2? This problem is NP-complete even if sets in C have cardinality 3.

Let $X = \{c_1, \ldots, c_m\}$. For the construction, we identify the elements of X with candidates and add an additional candidate x. For each set $\{c_i, c_j, c_k\} \in C$ with $i < j < k$ we introduce one vote: $\{c_i \succ c_j, x \succ c_k\}$. In addition, we add the vote $x \succ c_m \succ \cdots \succ c_1$. One can show that the resulting preference profile $\mathcal{P}$ is single-peaked if and only if (X, C) is a SET SPLITTING yes-instance. The key observation is that, on A, x separates the sets X_1 and X_2. □

It is important to note that – in contrast to the NP-hardness result in Theorem 5 – in this proof we make use of U-valleys instead of V-valleys. This means in particular that this hardness result does not hold for weak orders, which cannot contain U-valleys. This is not incidental: in the next section, we present a polynomial-time algorithm for weak orders.

5 The Guided Algorithm

In this section, we present a polynomial time algorithm for profiles of weak orders. This algorithm requires a total order to guide the placement of candidates on the axis. A (single) complete vote contained in the profile can be used for this task. We are able to weaken this constraint to requiring a so-called *guiding order*, which is a total order that is implicitly given in the profile.

Definition 7. A *guiding order* of a profile of partial orders is a total order $\langle c_1 < c_2 < \ldots < c_m \rangle$ on C with the following property: For each $i \in \{1, \ldots, m\}$ it holds that there is a vote $V \in \mathcal{P}$ such that c_i is the unique last ranked candidate in $V[\{c_i, c_{i+1}, \ldots, c_m\}]$.

Clearly, not all profiles of partial orders possess a guiding order. In particular the profiles constructed in the proof of Theorem 5 do not possess one.

It is easy to verify that $\mathcal{O}(m \cdot n)$ time suffices to verify whether a profile of weak orders contains a guiding order and, if so, to compute it. We thus consider the guiding order as part of the input.

Theorem 8. *Given a guiding order, the* WEAK ORDER SINGLE-PEAKED CONSISTENCY *problem can be decided in* $\mathcal{O}(m \cdot n)$ *time.*

We will refer to Algorithm 1, which Theorem 8 is based on, as the *Guided Algorithm.* Without loss of generality, we assume that the guiding order is $\langle c_1 < c_2 < \ldots < c_m \rangle$, i.e., we number the candidates based on the guiding order.

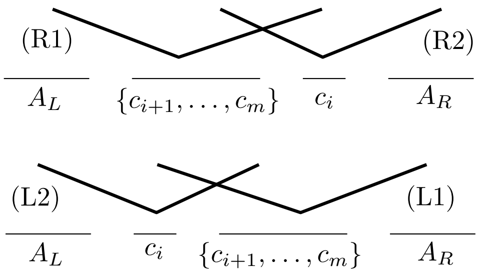

Figure 3: Graphical representation of the conditions testing whether c_i can be placed on the right-hand side ((R1), (R2)) or on the left-hand side ((L1), (L2))

The algorithm has a simple structure: The first (smallest) candidate in the guiding order is placed on the rightmost position of the axis (The leftmost position would work as well.) Starting with the second candidate in the guiding order, the candidates are successively placed on the axis – either at the leftmost or rightmost still available position. The lists A_L and A_R correspond to the left-hand and right-hand side of the axis. For each candidate, we test whether it can be placed on the right-hand side or left-hand side without creating a valley. If only one of these options is viable, the candidate is placed accordingly. If both left and right are possible, we place the candidate arbitrarily right. If neither is possible, the preference profile is not single-peaked.

Testing whether a vote V_k imposes restrictions on the placement of a candidate is achieved by four conditions. These conditions distinguish four categories of candidates: candidates in A_R, candidates in A_L, candidates that have not yet been placed ($C_{>i} = \{c_{i+1},\dots,c_m\}$) and the candidate that is currently under consideration (c_i). We are only checking for valleys that include c_i. This gives rise to these four conditions: (R1) and (R2) test whether placing c_i on the right-hand side of the already placed candidates leads to valleys, (L1) and (L2) do the same for the left-hand side. Refer to Figure 3 for a graphical representation. Since we only consider weak orders, we do not have to consider every candidate triple possibly fulfilling these conditions but have to check only maximal or minimal candidates. More specifically, checking whether there is a candidate $c \in A_L$ and $c' \in C_{>i}$ with $c \succ c'$ is equivalent to whether any maximal element in A_L is preferred to some minimal element in $C_{>i}$. Let $min_k(X)$ ($max_k(X)$) denote a function that picks an element that is minimal (maximal) with respect to $\succ_k$ in X.

$$c_i \succ_k min_k(C_{>i}) \text{ and } max_k(A_L) \succ_k min_k(C_{>i}) \qquad \text{(R1)}$$

$$max_k(C_{>i}) \succ_k c_i \text{ and } max_k(A_R) \succ_k c_i \qquad \text{(R2)}$$

$$c_i \succ_k min_k(C_{>i}) \text{ and } max_k(A_R) \succ_k min_k(C_{>i}) \qquad \text{(L1)}$$

$$max_k(C_{>i}) \succ_k c_i \text{ and } max_k(A_L) \succ_k c_i \qquad \text{(L2)}$$

Using these four definitions, we can give a succinct description of the algorithm (Algorithm 1). Theorem 8 claims that the Guided Algorithm requires $\mathcal{O}(m \cdot n)$ time. This is only possible if the conditions can be checked in constant time. Thus, the minima and maxima have to be computable in constant time. For $max_k(A_L)$ and $max_k(A_R)$ this is easily possible by storing and updating these two values. If c_i is placed left, we update $max_k(A_L)$ in case c_i is the new maximum (with respect to $\succ_k$); if c_i is placed right, we proceed analogously $max_k(A_R)$. For computing a minimal value of $C_{>i}$, observe that the set $C_{>i}$ becomes smaller with increasing i. Thus, a minimal value of $C_{>i}$ might disappear at some point and a new (larger) value has to be chosen. The new minimum is the smallest element (with respect to $\succ_k$) in $C_{>i}$ that is at least as large as the old minimum. The amortized cost of this update procedure is $\mathcal{O}(1)$. A maximal value of $C_{>i}$ can be found analogously.

Algorithm 1: The Guided Algorithm

```
1  $A_L \leftarrow \langle\rangle$; $A_R \leftarrow \langle c_1 \rangle$
2  for $i \leftarrow 2 \dots m$ do
3      right ← true; left ← true
4      for $k \leftarrow 2 \dots n$ do
5          if Condition (R1) or (R2) holds then
6              right ← false
7          if Condition (L1) or (L2) holds then
8              left ← false
9      if right = true then
10         $A_R \leftarrow \langle c_i < A_R \rangle$
11     else
12         if left = true then
13             $A_L \leftarrow \langle A_L < c_i \rangle$
14         else
15             return not_single_peaked
16 return $A_L < A_R$
```

The Guided Algorithm achieves a polynomial runtime due to two restrictions: it requires a guiding order and it is limited to weak orders. In the next section, we will consider profiles that do not have a guiding order.

6 The Unguided Algorithm

We now present a second polynomial-time algorithm (Algorithm 2). In contrast to the Guided Algorithm, this one is not dependent on a guiding order and we therefore refer to it as the Unguided Algorithm. The Unguided Algorithm is restricted to top orders. We also require the input preference profile to be *connected*: Let us consider the ranked candidates in a top order to be a hyperedge of a graph with candidates as vertices. A profile of top orders is called connected if this graph has only one connected component. This restriction does not limit the applicability: If two or more connected components exist in this graph, we can use the algorithm for each component (i.e., its respective candidates and votes) and concatenate the resulting axes in arbitrary order.

The algorithm works as follows: First, we choose a candidate c_{start} which is going to be the leftmost candidate on the axis A. Since we have no guiding order, each candidate might be at the leftmost position. Hence we loop over all candidates (Line 1). The corresponding axis under construction is $A = \langle c_{\text{start}} \rangle$. We now aim to complete this axis by adding candidates to the right in a way that all votes are single-peaked with respect to this axis. To this end we employ the loop in Line 3. In this loop (variable i) we infer from the already

Algorithm 2: The Unguided Algorithm

```
foreach $c_{start} \in C$ do
    $A \leftarrow \langle c_{\text{start}} \rangle$
    for $i \leftarrow 1 \ldots m$ do
        foreach $V \in$ VotesWithPeak$(a_i)$ do
            if $A \oplus V =$ incompatible then
                Continue with next $c_{\text{start}} \in C$ in Line 1.
            else
                $A \leftarrow A \oplus V$
        if $|A| = i$ and $i < m$ then
            $V \leftarrow$IntersectingVote$(A)$
            if $a_i \notin V$ then
                Continue with next $c_{\text{start}} \in C$ in Line 1.
            $x \leftarrow$ new candidate not in $C$
            $C' \leftarrow \{c \in V \mid c \succ a_i\} \cup \{a_i, x\}$
            $S \leftarrow \emptyset$
            for $k \leftarrow 1 \ldots n$ do
                $V'_k \leftarrow$ ReplTop$(V_k, C \setminus (A \cup C'), x)$
                $S \leftarrow S \cup \{V'_k[C']\}$
            $A' \leftarrow$GuidedSP$(S, V[C'], a_i, x)$
            if $A' =$not_single_peaked then
                Continue with next $c_{\text{start}} \in C$ in Line 1.
            else
                $A \leftarrow A \oplus A'[C' \setminus \{x\}]$
    return $A$
return not_single_peaked
```

placed candidate a_i (the i-th candidate on A from left) the candidate a_{i+1} (or even more candidates further to the right).

The Lines 4 to 8 are based on the following observation: Let us assume that at a certain point $A = \langle c_1 < c_2 < c_3 \rangle$ and $V = \langle c_3 \succ c_2 \succ c_4 \succ c_5 \rangle \in \mathcal{P}$. Since c_3, the peak of V, is already contained in A, there is only one compatible extension of A: $\langle c_1 < c_2 < c_3 < c_4 < c_5 \rangle$. We formalize this extension operation with the $\oplus$ operator:

Definition 9. Let A be an incomplete axis and V a top order. Furthermore, let $V[C \setminus A] = \langle c'_1 \succ c'_2 \succ \ldots \succ c'_j \rangle$. We define $A \oplus V = \langle A < c'_1 < c'_2 < \ldots < c'_j \rangle$ if V is single-peaked with respect to this axis and $A \oplus V =$ *incompatible* otherwise.

The loop in Line 4 enumerates all votes with peak a_i (VotesWithPeak(a_i)). Let $V \in$ VotesWithPeak(a_i). If $A \oplus V =$ *incompatible* then A cannot be extended to a complete (single-peaked) axis and we consider the next $c_{\text{start}} \in C$ in Line 1. Otherwise, we obtain a new incomplete axis $A \leftarrow A \oplus V$.

It might be the case that the candidate a_{i+1} has not yet been determined after these steps. The Lines 9 to 23 deal with this case. Since the election is connected there has to be at least one vote that contains both a candidate on A and a candidate that has not been placed yet. The procedure IntersectingVote in Line 10 returns such a vote V with $A \cap V \neq \emptyset$ and $V \setminus A \neq \emptyset$. Clearly, for such a vote V it holds that peak$(V) \notin A$. If this vote does not contain a_i, A cannot be extended to a single-peaked axis.

Now that we have such an intersecting vote V with $a_i \in V$, we employ the Guided Algorithm to find a further extension of A. The main idea is to use V as a guiding order and find an axis for the candidates in $\{c \in V \mid c \succ a_i\}$. In principle, this axis can be found independently of the existing axis A. However, the leftmost and rightmost candidates have to be chosen with regard to "external" considerations: The leftmost candidate has to be a_i, otherwise the axes A and A' could not be merged. For the rightmost candidate, we have to consider votes with candidates that are not being placed on the axis in this step. The following example illustrates the issue.

Example. Let $A = \langle c_1 \rangle$, $V_1 = \langle c_2 \succ c_3 \succ c_1 \rangle$ and $V_2 = \langle c_3 \succ c_4 \rangle$. The vote V_1 intersects A and hence $C' = \{c_1, c_2, c_3\}$. We employ the Guided Algorithm and might obtain $A' = \langle c_1 < c_3 < c_2 \rangle$. [1] Now observe that $A \oplus A' = A'$ can no longer be extended in a way that it is single-peaked for V_2. This would have been possible if c_3 had been chosen as the rightmost candidate in A'.

As we see from this example, we sometimes have to "force" the rightmost candidate in A'. We do this by adding an additional candidate x to every vote (Line 16 to 18). It is placed at the position of the highest ranked candidate in each vote that is not contained in $A \cup C'$. This is done by the ReplTop function: ReplTop(V, D, x) replaces the one candidate in vote V with candidate x that is the highest ranked of the candidates in D. By forcing this element X to be the rightmost candidate, we ensure that A' is chosen under consideration of all votes with ranked candidates not in C'.

Example (continued)**.** We apply ReplTop to the votes V_1 and V_2. The vote $V'_1 = \langle c_2 \succ c_3 \succ c_1 \rangle$ since ReplTop replaces a bottom candidate. The vote $V'_2 = \langle c_3 \succ x \rangle$. Now, we can only obtain the axis $\langle c_1 < c_2 < c_3 < x \rangle$.

The set S, as computed in Lines 15 to 18, is the election $\mathcal{P}$ restricted to C' together with the additional candidate x. We now employ GuidedSP$(S, V[C''], A', a_i, x)$ which means that we employ the Guided Algorithm for the profile S and guiding order $V[C']$. Furthermore, we require that the leftmost candidate on the axis is a_i and the rightmost is x. The function GuidedSP either returns not_single_peaked or an axis A'. If it returns not_single_peaked, the next $c_{\text{start}} \in C$ is considered (Line 1). Otherwise, we continue with the extended axis $A \leftarrow A \oplus A'$.

Theorem 10. *The* TOP ORDER SINGLE-PEAKED CONSISTENCY *problem can be solved in $\mathcal{O}(m^2 \cdot n)$ time.*

Proof. The main loop (Line 1) iterates over all m candidates. The loop in Line 4 iterates over every vote at most once. Consequently, the $\oplus$ operator is applied at most $m \cdot n$ times. Since $A \oplus V$ can be computed in $\mathcal{O}(m)$ time, the Lines 4 to 8 have a total runtime of $\mathcal{O}(m^2 \cdot n)$.

It remains to determine the runtime of the Lines 9 to 23. Precomputation of the IntersectingVote procedure allows its execution in $\mathcal{O}(m)$ time. This is achieved by storing two votes for each candidate $c \in C$. One of the two

[1]Whether we obtain this axis or $\langle c_1 < c_2 < c_3 \rangle$ depends on whether the algorithm prefers placing candidates to the left or to the right if both choices are possible.

votes for c is an intersecting vote for any axis that has c at its rightmost occupied position (assuming single-peakedness). To find these two votes consider the set of votes for which the sets $\{c' \in C \mid c' \succ c\}$ are maximal (with respect to $\subseteq$). If we consider a single-peaked axis, then candidates in such a set have to form a contiguous subsequence either directly left or directly right of c. Since these sets are maximal, only two of them can exist. Consequently, we compute these maximal sets for each candidate. If three or more exist for one candidate, we can terminate the algorithm already at this point. Also, if two maximal sets have a non-empty intersection, the algorithm terminates. (The candidates in the intersection would have to lie both left and right of c). This precomputation step requires $\mathcal{O}(m^2 \cdot n)$ time. However, the `IntersectingVote` procedure only has to choose from two votes and thus requires only $\mathcal{O}(m)$ time.

The set S can be generated in $\mathcal{O}(|C'| \cdot n)$ time, as well as applying `GuidedSP` (Theorem 8). Observe that after applying the Guided Algorithm, the candidates in C' are placed on the axis. Consequently, the Guided Algorithm is applied to sets of candidates that overlap by at most one candidate. Hence for a fixed $c_{\text{start}} \in C$ the total runtime of all calls of the Guided Algorithm is in $\mathcal{O}(m \cdot n)$. Taking the main loop into account, we obtain a total runtime of $\mathcal{O}(m^2 \cdot n)$. □

7 Experiments

In this section we want to demonstrate the practical applicability of our algorithms. For this purpose we have implemented both the Guided Algorithm and the Unguided Algorithm in Java. We have generated random single-peaked elections with $n = 100,000$ (votes) and $m = 10, 50, 100, 250, 500, 750, 1000$ (candidates). These have been generated by first picking an arbitrary axis and then randomly generating total orders that are single-peaked with respect to this axis. In these total orders we have removed information to introduce incomparability: For generating weak orders, we have made adjacent candidates in the total orders incomparable to one another with probability p. Testing the Guided Algorithm implementation has shown that this probability p has no significant impact on the runtime and thus we have chosen p to be 0.5. For generating top orders, we have kept the top r candidates in each vote and made the remaining candidates incomparable to one another. We have generated top orders with (a) $r = 10$ and (b) $r = m/3$. Random instances (which are likely not single-peaked) can be solved much faster and are thus not considered here.

The runtimes (averaging 5 instances for every measuring point) are displayed in Figure 4. The benchmarks have been performed on a single core of an Intel Core i5-3320M CPU with 2.60GHz using OpenJDK 7. The time required to copy the input to the RAM is not included. (This would have distorted the measurements since the input files had a size of up to 370 MB.)

One can observe that the Unguided Algorithm has an approximately quadratic runtime in (b) but an approximately linear runtime in (a). This is because the `GuidedSP` procedure is less likely to be required for large instances. Hence only the runtime of the first part of the algorithm dominates,

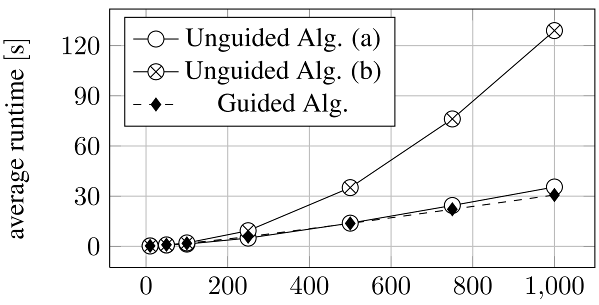

Figure 4: The impact of the number of candidates

which is $\mathcal{O}(m \cdot n \cdot r)$. Thus for $r = 10$ (a) we can observe linear runtime and for $r = m/3$ (b) quadratic runtime. Our experiments also confirmed that the number of votes has a linear impact on the runtime – this is, however, not shown in the figure. To sum up, these benchmarks demonstrate that our algorithms can efficiently handle even large instances with 1,000 candidates and a large number of votes.

8 Conclusions

In this paper we have analyzed the PARTIAL ORDER SINGLE-PEAKED CONSISTENCY problem. Despite its NP-completeness for partial orders in general, we have found two fast algorithms for plausible application scenarios. The Guided Algorithm requires a guiding order as additional input. Such an order is likely to exist for large preference profiles. In the case that top orders are elicited, a guiding order might not exist. Here the Unguided Algorithm is applicable.

We would like to mention one particular application of the Unguided Algorithm concerning single-peaked scoring protocols. Scoring protocols are specified by a scoring vector $(\alpha_1, \ldots, \alpha_m)$. A vote $V = \langle c_1, \ldots, c_m \rangle$ gives α_1 points to c_1, α_2 points to c_2, etc. The winner candidate is determined by summing over all votes. Often scoring vectors of the type $(\alpha_1, \ldots, \alpha_k, 0, \ldots, 0)$ with $\alpha_1 > \ldots > \alpha_k > 0$ are considered. For such scoring rules, top orders (with k ranked candidates) constitute *full* information. It is therefore debatable whether the input may be considered to be given as a profile of total orders. This is of relevance for single-peaked profiles. For example, Brandt *et al.* [2010] study the constructive coalition weighted manipulation problem for scoring protocols in single-peaked elections. The authors consider the axis to be part of the input (for good reasons as explained in their paper). The computation of such an axis with existing algorithms is possible only if preferences are specified by total orders and thus contain problem-irrelevant information. If only relevant information is given, i.e., the input consists of top orders, an algorithm such as the Unguided Algorithm is required.

The work in this paper can be extended in several directions. Exemplarily, we want to mention recent results on nearly single-peaked profiles [Faliszewski *et al.*, 2011a; Cornaz *et al.*, 2012; Erdélyi *et al.*, 2013; Elkind *et al.*, 2012; Bredereck *et al.*, 2013; Sui *et al.*, 2013]. Notions of nearly single-peakedness are robust and applicable in real-life settings. Extending the algorithms presented in this work to nearly single-peaked domains is thus of high relevance.

References

[Bartholdi and Trick, 1986] John J. Bartholdi and Michael Trick. Stable matching with preferences derived from a psychological model. *Operations Research Letters*, 5(4):165 – 169, 1986.

[Baumeister and Rothe, 2012] Dorothea Baumeister and Jörg Rothe. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Information Processing Letters*, 112(5):186 – 190, 2012.

[Baumeister *et al.*, 2012] Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. Campaigns for lazy voters: truncated ballots. In *Proc. of AAMAS-12, Volume 2*, pages 577–584, 2012.

[Betzler and Dorn, 2010] Nadja Betzler and Britta Dorn. Towards a dichotomy for the possible winner problem in elections based on scoring rules. *Journal of Computer and System Sciences*, 76(8):812–836, 2010.

[Black, 1948] Duncan Black. On the rationale of group decision making. *Journal of Political Economy*, 56(1):23–34, 1948.

[Brandt *et al.*, 2010] Felix Brandt, Markus Brill, Edith Hemaspaandra, and Lane A. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. In *Proc. of AAAI-10*, pages 715–722, 2010.

[Bredereck *et al.*, 2013] Robert Bredereck, Jiehua Chen, and Gerhard J. Woeginger. Are there any nicely structured preference profiles nearby? In *Proc. of IJCAI-13*, 2013.

[Conitzer, 2009] V. Conitzer. Eliciting single-peaked preferences using comparison queries. *Journal of Artificial Intelligence Research*, 35:161–191, 2009.

[Cornaz *et al.*, 2012] Denis Cornaz, Lucie Galand, and Olivier Spanjaard. Bounded single-peaked width and proportional representation. In *Proc. of ECAI-12*, volume 242 of *FAIA*, pages 270–275. IOS Press, 2012.

[Doignon and Falmagne, 1994] Jean-Paul Doignon and Jean-Claude Falmagne. A polynomial time algorithm for unidimensional unfolding representations. *Journal of Algorithms*, 16(2):218–233, 1994.

[Dwork *et al.*, 2001] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. of WWW-01*, pages 613–622. ACM, 2001.

[Elkind *et al.*, 2012] Edith Elkind, Piotr Faliszewski, and Arkadii M. Slinko. Clone structures in voters' preferences. In *Proc. of EC-12*, pages 496–513. ACM, 2012.

[Erdélyi *et al.*, 2013] Gabor Erdélyi, Martin Lackner, and Andreas Pfandler. Computational aspects of nearly single-peaked electorates. In *Proc. of AAAI-13*. AAAI Press, 2013.

[Escoffier *et al.*, 2008] Bruno Escoffier, Jérôme Lang, and Meltem Öztürk. Single-peaked consistency and its complexity. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, volume 178 of *FAIA*, pages 366–370. IOS Press, 2008.

[Fagin *et al.*, 2003] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.

[Fagin *et al.*, 2006] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.

[Faliszewski *et al.*, 2011a] Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. The complexity of manipulative attacks in nearly single-peaked electorates. In *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-11)*, pages 228–237, 2011.

[Faliszewski *et al.*, 2011b] Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Information and Computation*, 209(2):89–107, 2011.

[Hemaspaandra *et al.*, 1997] Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

[Hemaspaandra *et al.*, 2005] Edith Hemaspaandra, Holger Spakowski, and Jörg Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[Konczak and Lang, 2005] Kathrin Konczak and Jerome Lang. Voting procedures with incomplete preferences. In *In Proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, 2005.

[Opatrny, 1979] Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.

[Pini *et al.*, 2011] Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Incompleteness and incomparability in preference aggregation: Complexity results. *Artificial Intelligence*, 175(7-8):1272 – 1289, 2011.

[Sui *et al.*, 2013] Xin Sui, Alex Nienaber, and Craig Boutilier. Multi-dimensional single-peaked consistency and its approximations. In *Proc. of IJCAI-13*, 2013.

[Walsh, 2007] Toby Walsh. Uncertainty in preference elicitation and aggregation. In *Proc. of AAAI-07*, pages 3–8. AAAI Press, 2007.

[Xia and Conitzer, 2011] Lirong Xia and Vincent Conitzer. Determining possible and necessary winners under common voting rules given partial orders. *Journal of Artificial Intelligence Research*, 41(2):25–67, 2011.