Algorithms and Experiments for (Nearly) Restricted Domains in Elections

Tomasz Przedmojski

February 4, 2016

TU Berlin

Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Chair Algorithmics and Computational Complexity

Algorithms and Experiments for (Nearly) Restricted Domains in Elections

Tomasz Przedmojski

1. Reviewer Prof. Dr. Rolf Niedermeier
Electrical Engineering and Computer Science
TU Berlin

2. Reviewer Prof. Dr. Sabine Glesner
Electrical Engineering and Computer Science
TU Berlin

Supervisors Prof. Dr. Rolf Niedermeier, Dr. Robert Bredereck and Dr. Jiehua Chen

February 4, 2016

Tomasz Przedmojski
Algorithms and Experiments for (Nearly) Restricted Domains in Elections
February 4, 2016
Reviewers: Prof. Dr. Rolf Niedermeier and Prof. Dr. Sabine Glesner
Supervisors: Prof. Dr. Rolf Niedermeier, Dr. Robert Bredereck and Dr. Jiehua Chen

TU Berlin
Chair Algorithmics and Computational Complexity
Institute of Software Engineering and Theoretical Computer Science
Electrical Engineering and Computer Science
Ernst-Reuter-Platz 7
10587 Berlin

Abstract

In this master thesis we implement algorithms for detection of single-peaked or single-crossing preference profiles, and construction of nearly single-peaked or single-crossing profiles by deleting candidates or voters from an election. We also contribute our own algorithm for SINGLE-PEAKED CANDIDATE DELETION which has better time-complexity than the established algorithm. We employ Algorithm Engineering method to guide the implementation process. We use the implemented algorithms to evaluate the real-world election data available from PrefLib. We find no single-peaked or single-crossing preference profiles in PrefLib and limited evidence for nearly restricted domains in two of the eight data sets available from PrefLib.

Abstract (German)

In dieser Masterabiet implementieren wir Algorithmen für die Erkennung von single-peaked oder single-crossing Präferenzprofilen und für die Konstruktion von single-peaked oder single-crossing Präferenzprofilen durch Löschen von Kandidaten oder Wähler aus einer Instanz von Wahlen. Unser weiterer Beitrag besteht in einem Algorithmus für SINGLE-PEAKED CANDIDATE DELETION mit einer besseren Zeitkomplexität als der etablierte Algorithms. Wir benutzen die Algorithm-Engineering-Methode um die Algorithmen zu implementieren. Wir setzen die implementieren Algorithmen ein, um die echten Daten aus PrefLib zu untersuchen. In unserer Analyse finden wir keine single-peaked oder single-crossing Präferenzprofile und wenige Beweise, in zwei von acht Datensätzen aus PrefLib, für Vorhandensein von Präferenzprofilen, die single-peaked- oder single-crossing-ähnlich sind.

Contents

Introduction 1

In this master thesis we aim to analyse and categorise real world election data provided by PrefLib [MW13] to asses whether and how often the idealised models of single-peaked and single-crossing elections actually occur in practical applications. Furthermore we try to answer the question of how different is the data from the single-peaked and single-crossing domains where applicable. Our work falls into the field of computational social choice

This chapter provides an overview of our work on restricted domains in elections within computational social choice. We first provide a general introduction into the problem using simple explanations and examples. Following this, we describe a more general context and interdisciplinary applications of the knowledge gained through this work. After that we discuss the content and the structure of this document and the main contributions that we have made. Finally we provide a summary of our results and provide some insight and hints for possible extensions and further work in related fields.

1.1 Motivating Example

For an introductory example, meet the four close friends: Adam, Betty, Caroline and Dominick who want to actively spend an afternoon together doing sports. Each of them has their own preferences regarding the sports they like and dislike.
Adam is most fond of archery, since he has trained with his father since childhood and loves sharing his passions with his friends.
Betty, having a very small frame and little weight to her person finds this idea interesting but a bit scary, and would prefer to do something else. In her mind the ideal choice would be to go bicycling.
Caroline on the other hand despises violence and everything associated with it. Still, she promised to Adam to try archery with him one day, but on this particular afternoon she would rather go canoeing, especially since the weather is sunny, without wind our clouds.
This does not sit well with Dominick at all, having went canoeing with Caroline multiple times in the past, the vivid memories of their friendship barely surviving each trip keep his opinion firmly against the idea. Bow shooting on the other hand sounds exciting to him, and thus more attractive than cycling. All in all it is just like

darts, and Dominick loves darts. Has he already said that he loves darts? Sadly his friends (with perhaps the exception of Adam) do not share his passion. Because the group started to get confused with multitude of options and who likes which option best they decided to draw a table with the first, second, third and fourth choices of each participant. They've been kind enough to share this table with us in Fig. 1.1. It should be noted that Dominick was adamant on putting darts and his first and second choice, but his friends managed to convince him otherwise.

	First choice	Second choice	Third choice	Fourth Choice
Adam	Archery	Darts	Canoeing	Bicycling
Betty	Bicycling	Archery	Canoeing	Darts
Caroline	Canoeing	Darts	Bicycling	Archery
Dominick	Darts	Bicycling	Archery	Canoeing

Fig. 1.1: Preferred activities on one sunny afternoon for a group of four friends. Note that putting the same option twice was disallowed.

Adam, a student of computer science, noticed that nobody shares each other first choices, and wanting to impress his friends with his knowledge of history (in this particular case gained only accidentally through the study of his native field), proposed that they use the Condorcet method from 1785 to agree on the afternoon activity. "Condorcet method goes as follows: we compare each option against every other option. We who among us would prefer one to the other. The option that wins every pairing is the one preferred against every other alternative and should most fairly win.".

Adam, Betty, Caroline and Dominick start the election and compare archery against bicycling, archery wins. They follow with archery against darts, archery again arriving ahead of the alternative. With Adam grinning they proceed to compare archery with canoeing and archery wins again. Adam's triumph seemed near but then archery lost to bicycling. After few more rounds they found out that bicycling tied with darts. Adam's friends then became impatient with him, and almost to spite him, decided as a majority to go bicycling.

What Adam and his friends faced is one of the oldest known problems in social choice: finding out the Condorcet winner of an election. Condorcet winner has a number of desirable properties like being the unique winner and aligning with an intuitive notion of what is fair and works best for the majority of people. Sadly even though a Condorcet winner is guaranteed to be unique, it does not exist for every election – just as Adam, Betty, Caroline and Dominick found out. What they missed is that there was a so called weak Condorcet winner: an option that beats or ties with every other option. By their definitions there can be multiple weak Condorcet winners but only one strong Condorcet winner. In this case the weak Condorcet winner was archery it won against every other option but bicycling and tied with bicycling (bicycling lost to darts, hence there was only one weak Condorcet winner). Adam wished he had remembered this as read on social choice during the evening.

On the next occasion the friends decided to do something less prone to conflict and made plans to go to see a movie. Betty, Caroline and Dominick were ashamed of their lack of patience with Adam and their pettiness and proposed to give Condorcet another go in an effort to redeem Adam. First they had to decide on the movie and pick from Star Wars, Spectre, The Revenant, and The Hateful Eight. They proceed to draw a table just as before and just as before they have kindly decided to share it with us in Fig. 1.2.

	First choice	Second choice	Third choice	Fourth Choice
Adam	Star Wars	The Hateful Eight	Spectre	The Revenant
Betty	Spectre	Star Wars	The Revenant	The Hateful Eight
Caroline	The Revenant	Spectre	Star Wars	The Hateful Eight
Dominick	Star Wars	Spectre	The Revenant	The Hateful Eight

Fig. 1.2: Movie preferences for a group of four friends.

This time the group has found a Condorcet Winner: Star Wars. It beats Spectre, The Hateful Eight and The Revenant in pairwise comparison. This choice seems fair to everybody and the fair in the Condorcet method has been restored, so that the group wants to give it one more try. As Star Wars was a highly anticipated film, it is screened several times a day and the group has to agree on the exact time. The film runs four times a day in their favorite cinema: at 16:00, at 17:00, at 18:00 and at 20:00. The group again draws preference profiles as a table shown in Fig. 1.3.

	First choice	Second choice	Third choice	Fourth Choice
Adam	20:00	18:00	17:00	16:00
Betty	20:00	16:00	18:00	17:00
Caroline	16:00	17:00	20:00	18:00
Dominick	16:00	17:00	18:00	20:00

Fig. 1.3: Screening time preferences for a group of four friends.

The third attempt at Condorcet method has proven unsuccessful at first: no winner could be determined since 20:00 and 16:00 tied. This has given Adam the opportunity to tell the group about weak Condorcet winners, which may have turned the tides in favor of seeing the movie at 20:00.

Unbeknownst to them the group of friends went on a journey through the fields of computational social choice, touching on politics, economy, psychology and sociology when they decided to use this strict and formal approach to picking an option.

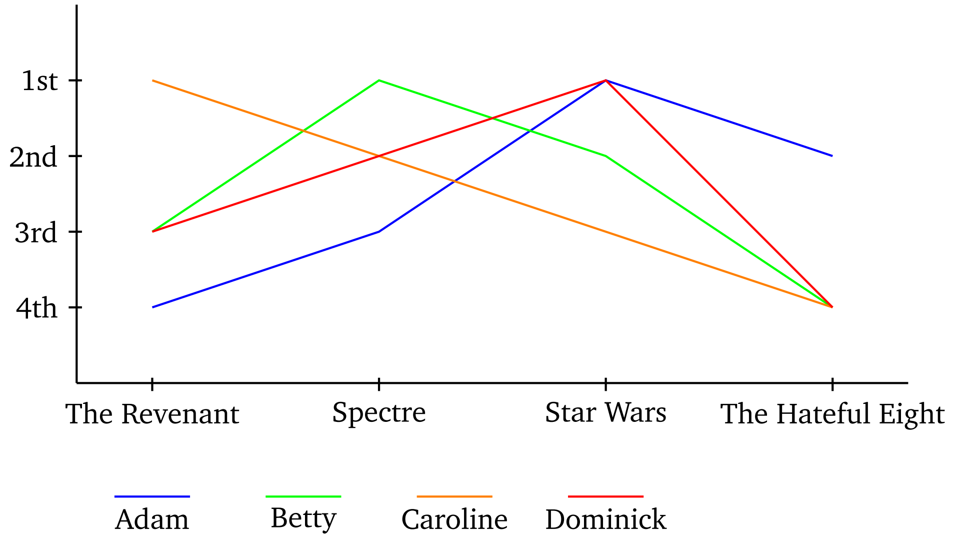

Fig. 1.4: Movie preferences of the four friends constitute a single-peaked preference profile. On the Y-axis we have plotted the positions in the ranking, and on the x-axis we have put the candidates.

In each case they have faced an election: they were group of voters which had to agree on a candidate (activity, movie, time). They did it by revealing their preferences through ranking and applying an election method (or algorithm) to it. The preference profiles here have been complete and strict: they contained every candidate exactly once, without ties.

Furthermore they have created two special preference profiles: single-peaked and single-crossing profiles. Their movie choices have been single-peaked and time preferences single-crossing. This special classes of profiles are examples of restricted domains. A restricted domain contains elements from a domains restricted through some additional constraints (besides the ones which a standard for a given domain). One could argue that a restricted domain is just another, smaller domain, but this obfuscates the whole point of using the restricted domains: to gains new insight into the original problem by looking at a part of it with special preferences.

What is a single-peaked preference profile exactly? Single-peaked preferences are special in that allow us to order the candidates on an axis ("from left to right") to have a very interesting property: if we go from left to right, we first encounter candidates which are better then the ones encountered before, then we reach a peak candidate and after that the candidates get worse and worse. Note that it possible to have multiple peak candidates (in most extreme case each voter can have it's own peak candidate). To ease understanding of single-peaked profiles we have provided a drawing of single-peaked profile (movie preferences) in Fig. 1.4 with an emphasis on *peaks*.

In Fig. 1.4 we actually have three peaks: Spectre and Star Wars are the obvious ones and the one less obvious is The Revenant. For The Revenant the "left side" of the peak is missing, that is there are no candidates to the left of The Revenant. This is allowed by the definition, even if it may seem a little bit counter-intuitive at first. It should be obvious by now that the peaks are also the top candidates (i.e. the most preferred) for each voter, therefore the lines depicting Adam's and Dominick's profile meet at the peak of Star Wars.

In contrast to movie preferences, the screening time preference profile is not single-peaked, but it is single-crossing. Single-crossing preferences are similar in spirit to single-peaked preferences and allow us to order the *voters* on an axis so that the axis can be split in two consecutive parts for every candidate pair. This partitioning has one additional constraint: one part contains candidates which prefer one candidate to the other and the second part has exactly the opposite preference. Therefore for every candidate pair we can pinpoint the point on the axis where the preference of the following voters changes regarding this point, that is we are able to identify the crossing point.

In Fig. 1.5 we plotted the single-crossing preference profile (screening time preferences for the four friends) and marked crossing points for every candidate pair. Blue line in the diagram spans voters which prefer 20:00 in a given comparison, 18:00 uses green, 17:00 uses orange and 16:00 uses red. So for example everyone but Dominick prefer 20:00 to 18:00, and everyone but Adam prefer 16:00 to 17:00 and 16:00 to 18:00. For the pairs 20:00 vs 17:00, 20:00 vs 16:00, and 18:00 vs 17:00 the four friends can be split into two factions: Adam and Betty supporting the first screening time of each pair, and Caroline and Dominick preferring the second screening time.

Single-crossing and single-peaked preferences have a number of interesting properties among which there is the following guarantee: if a preference is single-peaked or single crossing, then there is at least one weak Condorcet winner[Cor+13].

1.2 Importance of Single-Peaked and Single-Crossing Profiles

Until now both single-peaked and single-crossing preference profiles may seem like a gimmick, examples of restricted domains which may emerge in real-world scenarios. But an excursion into other scientific disciplines will reveal how important and how broad are the applications and consequences of these restricted domains.

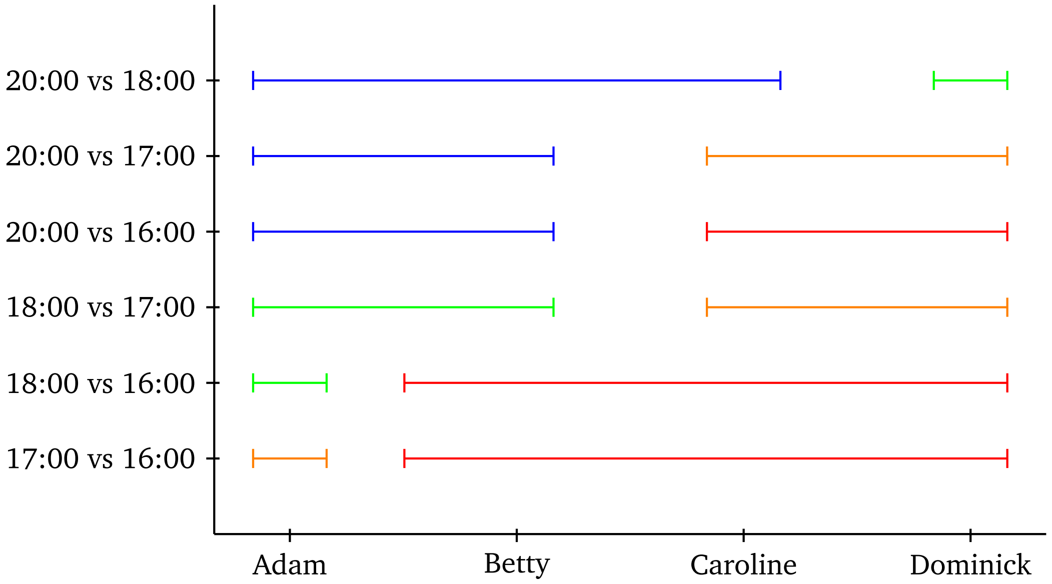

Fig. 1.5: Screening time preferences of the four friends constitute a single-crossing preference profile. On the Y-axis we have plotted each candidate pair, and on the x-axis we have put the candidates. For each pair we have marked which voters prefer which candidate from a given pair.

The history of importance of single-peaked preferences begins with political science and political economy, as this restriction is used for the proof of median voter theorem as shown by one of the fathers of social choice, theory Duncan Black, in his 1948 paper "On the Rationale of Group Decision-making" [Bla48] which was later expanded upon by Anthony Downs in an extremely influential book "An economic theory of political action in a democracy" [Dow57]. The two then collaborated on another influential book "The theory of committees and elections" [Bla+58].
But the book that probably established social choice theory as independent field of research is the Kenneth Arrow's seminal "Social Choice and Individual Values" [Arr51] which among research on single-peaked preferences contains "General Possibility Theorem", better known as "Arrow's impossibility theorem".

Single-peaked preferences also play an important role in economics, dealing with the (Fair) Division Problem [Spr91; Tho94] and the Uniform Rule [Chi94; Ott+96].

In psychology single-peaked preferences have been linked to the Theory of Preference [CA77b]. Single-peaked preferences have also been used in mathematical psychology [Coo69; CA77a; Asc78; Asc81].

Also in the field of computational social choice single-peaked preferences play an important role. For example, the Kemeny-Young method, which can be seen as more sophisticated and extended version of the Condorcet method, is NP-hard in general, but can be computed in polynomial time for single-peaked preference

profiles [Cor+13].
In a similar spirit the typically hard problems of manipulation [BI+89], control [Bar+92], and bribery [Fal+06] in many election schemes become polynomial time solvable when dealing with single-peaked preferences [Fal+09; Bra+15].

Single-crossing preferences may have a slightly less impressive track record but are nevertheless important in many disciplines. They are introduced in Roberts' 1977 paper on "Voting over income tax schedules" [Rob77], which is a major contributor to the understanding of optimal taxation.

In economics and social choice, single-crossing preferences have been linked to various aspects of majority rule [Gra78] [Rot90] [GS96], strategic voting [ST06] [Sap09], coalition formation [Dem94] [KUN06], income redistribution [MR81] local public goods and stratication [Wes77] [EP98].

Similarly to single-peaked preferences, single-crossing preferences allow to compute Kemeny rankings in polynomial time [Cor+13].
Another parallel between the two emerges when considering election control problems under single-crossing preferences, which then become polynomial-time solvable [MF14].

From this brief overview, which is in no way complete, one can deduce that the question of relevance of the single-peaked and single-crossing preference profiles in the real world is not only interesting to complexity-theory oriented computer scientists within computational social choice, but also has implications for many disciplines of science.

1.3 Problem Statement

After having introduced single-peaked and single-crossing profiles and having briefly discussed their importance in many disciplines, one may be inclined to ask how often do these restricted domains occur in real life?
From a purely combinatorial perspective both restrictions should occur very infrequently. Based on this one may further ask: how much different are the preference profiles from single-peaked and single-crossing profiles?

In this work we aim to answer these questions for the data sets provided by PrefLib. PrefLib [MW13] is a collaborative effort of scientists from many disciplines (ranging from computer science to social sciences) dealing with elections, preferences, and matchings to assemble a database of real-world data which can be used for research.

This project is lead by Nicholas Mattei and Toby Walsh from the University of New South Wales.

PrefLib provides a multitude of data sets from many different applications: rankings, election results, matchings and more. We will be dealing exclusively with election data with strict and complete preference profiles.

To state it more formally: we will be evaluating the data sets provided by PrefLib to find out the minimum distances of the preference profiles to single-peaked and single-crossing preference profiles. We will be using two most intuitive distance measures: minimum number of candidates that have to be deleted to reach a restricted profile and minimum number of voters that have to be deleted to reach a restricted profile.

To this end, we will implement, optimise, and extend existing algorithms and approaches, but also contribute algorithms and approaches of our own. One of our most important tools in this work is the Algorithm Engineering approach.

To the best of our knowledge, this is the first time the evaluation of (nearly) single-peaked and (nearly) single-crossing profiles has been performed for the data gathered in PrefLib.

1.4 Thesis Structure

This document contains five chapters, each describing a logically isolated aspect of our work. This structure does not reflect the actual order in which we did our work (we worked iteratively, one algorithm at a time), but provides a much better insight and a clear narrative for people not extensively familiar with the subject.

What follows is the brief summary of each chapter and how it fits into the thesis.

Chapter 1

The first chapter contains the introduction and some general perspective. We use informal descriptions and lively examples to show concepts and problems. Furthermore it contains the abbreviated and summarised results of our work and recommendations regarding further research and applications.

Chapter 2

In the second chapter we present the current state of knowledge of the studied field. We provide established mathematical definitions for crucial terms like elections, preferences, preference profiles, weighted preferences, single-peaked profiles, and single-crossing profiles. We also discuss the limitations of the chosen models.

With basic tools and terms introduced, we proceed to present a different perspective on single-peaked and single-crossing restrictions: through forbidden substructures, which are then used in presented algorithms.

We then continue to review and discuss the currently known algorithms and approaches to detecting single-peaked or single-crossing preference profiles, constructing axes for single-peaked or single-crossing profiles, and computing distances of a preference profile to single-peaked or single-crossing measured by a number of deleted candidates or measured by the number of deleted voters.

We also briefly touch on the computational complexity of the problems related to single-peaked and single-crossing profiles, because two of them (SINGLE-PEAKED MAVERICK DELETION, SINGLE-CROSSING CANDIDATE DELETION) are not polynomial-time solvable but rather NP-hard.

This review established the context and formal definitions for the following chapters.

Chapter 3

The third chapter deals with our theoretical contribution in the field of computational social choice. We present and prove a $\mathcal{O}(m^3n)$ algorithm (where m is the number of candidates and n is the number of pairwise disjoint preference profiles) for computing the minimum distance of a preference profile to a single-peaked preference profile, measured by the number of deleted candidates (SINGLE-PEAKED CANDIDATE DELETION). We also introduce worst candidate sets as a tool of reasoning about nearly single-peaked profiles.

Our contribution builds-on an existing algorithm with both higher time complexity $\mathcal{O}(m^5n)$ and higher space complexity, which has proven impossible to implement for us. We briefly discuss our difficulties and reasons for implementing our own algorithm.

Our algorithm is based on dynamic programming. We give two formulations: one more typical top-down mathematical perspective, and a bottom-up pseudo-code formulation and write-up which is more suitable for an actual efficient implementation on typical hardware. We prove the correctness of our algorithm by induction.

We complete the chapter by discussing some interesting aspects of the algorithm, which came into the play during the analysis and the implementation.

Chapter 4

In the fourth chapter we use the Algorithm Engineering method to successfully develop and deploy the algorithms required for the six problems described in Chapter 2 which central to this thesis.

We begin the chapter with a brief overview of the Algorithm Engineering method and discuss how we leveraged its different elements dealing with different challenges.

As a natural starting point for the Algorithm Engineering approach we give the description of our target setup, the benchmarking and testing conditions, and also constraints. We provide the reasoning behind our choices and discuss their consequences.

Next, we discuss the process of implementing the single-peaked detection algorithm, the challenges and the pitfalls that we have faced, and tools used to deal with them. We follow with a similar discussion for the single-crossing detection algorithm.

Then we proceed to a report on the two polynomial-time algorithms for distances, focusing especially on the tricky aspects of the nontrivial algorithms and means used to ensure the correctness of the implementation.

After that we describe at length our approach to dealing with the remaining two NP-hard problems. We emphasise the importance of the Algorithm Engineering method to achieving a success in our case. We contrast our final solution to the starting point which is based on the current domain knowledge. Possible natural extensions and continuations are provided based on the related fields.

We conclude the chapter with an evaluation of our approach, how well it performed, and how closely it matched the idealised method.

Chapter 5

The final, fifth chapter of this thesis contains the extensive evaluation of the data sets. We inspect the data provided by PrefLib to find whether it contains single-crossing or single-peaked instances. After that we measure distances for the remaining instances, first by trying to delete the candidates and then by deleting the mavericks (voters).

The chapter also contains the description of the data sets and comments on how they logically and thematically belong together. Thus the analysis is performed both in aggregate and for the isolated data sets.

We present multiple interpretations and visualisations and answer multiple interesting questions regarding the data.

We finish the chapter and our thesis by commenting on the results and providing some insight and recommendations for further work.

1.5 Results and Future Work

In this thesis we have analysed 314 instances (election data) provided by PrefLib (as of February 4, 2016 it was the complete set). These instances have been grouped in 8 data sets. Altogether we found out that no instances have been single-crossing or single-peaked. Furthermore, only for two data sets we found some evidence for nearly single-peaked and nearly single-crossing domains.

When dealing with the NP-hard problems SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING CANDIDATE DELETION we were able to solve 99% and 75% of the instances given the constraints of our methodology (see Sections 4.2, 4.5 and 5.1).

Our theoretical contribution is an efficient $\mathcal{O}(m^3 n)$ (where m is the number of candidates and n is the number of pairwise disjoint preference profiles) algorithm for finding a minimum set of candidates which need to be deleted to obtain a single-peaked preference profile.

We can identify four natural extensions of our work. The first is the most intuitive one and involves obtaining more data sets from different science disciplines and evaluating them similarly to our approach.
The second involves a natural extension or our tools: the inclusion of additional distance measures. From our point of view the most interesting distance measures would be a distance measure that combines candidate and voter (maverick) deletion. This is a very hard problem, as it requires an optimisation of multiple criteria.
The third extension would focus more on the algorithmic side of our work and improve the practical solutions for NP-hard problems SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING CANDIDATE DELETION. Such improvements can be made by borrowing from the research on related, better understood problems, like the Hitting-Set problem.

The final extension would require significantly more resources than a master thesis permits, but would also have a higher impact: extending the analysis to include partial orders, possibly with ties. The number of instances of this type available in PrefLib is two orders of magnitude greater than for strict complete orders. The intuitive reason as to why such work would be resource intensive stems from the fact that even deciding whether an incomplete profile is single-peaked is NP-complete [Lac14]. The same is also true for single-crossing preferences [Elk+15].

Preliminaries 2

In this chapter we provide an overview of the terms, definitions, and algorithms relevant to our thesis. Formal definitions are supplemented by informal descriptions and illustrated by examples, where suitable.

We begin by defining the most basic terms including preferences, candidates, voters, elections. Then we proceed to formal definitions of single-peaked and single-crossing preference profiles. We discuss multiple formulations which provide different perspectives on the respective problems.

We focus on formal definitions of two distance measures which are used throughout this thesis. The two distance measures are as follows: deleting a number k of candidates so that a preference profile in an election without these candidates is single-peaked (or single-crossing), and deleting a number k of voters so that a preference profile without these voters is single-peaked (or single-crossing). Since we are dealing with two types of restricted domains we arrive at four problems which we formally define; we briefly touch on the computational complexity of these four problems.

We continue our review with six algorithms which play a crucial role in achieving our goal as stated in the first chapter. We first deal with two algorithms that detect single-peaked and single-crossing profiles, but also return the candidate and the voter axes. We then proceed to discuss two polynomial-time algorithms for single-peaked candidate deletion and single-crossing maverick (voter) deletion. Finally, we reach the NP-hard problems single-peaked maverick-deletion and single-crossing candidate deletion and describe approaches to dealing with these problems.

2.1 Definitions

An election is simply a pair of two sets: a set of candidates and a set of voters. They have wide variety of applications in multiple fields from many disciplines (AI research, autonomous systems, distributed systems, game theory, voting systems and many more), most of which share a seemingly simple goal: to pick one or more candidates from a set. A set of candidates can contain anything from real political candidates to movies, appointments, database servers, any set of subjects really. Each voter in the voter set has a preference order over the candidate set. In this thesis, we assume that each of the preference orders is a linear order over the candidate set. Depending on the context and on the exact scenario, we sometimes use the terms voter and preference interchangeably, depending which one feels more natural to us.

In this thesis, we deal exclusively with a very simple and strict form of preferences: strict total orders. A strict total order is a linear order which is transitive, asymmetric and total [NK04]. A set (or a multiset of) of preferences in an election is called preference profile.

Definition 2.1 (Preference relation)**.** A *preference relation* v on C is a strict total order on C.

Definition 2.2 (Election)**.** A pair $E = (C, V)$ with $|C| = m$ and $|V| = n$ is called *an election* if C is a nonempty, finite set of candidates and V is a set of preference relations on C.

In certain situations an election may contain duplicate preferences, i.e. there can be some preferences that occur multiple times. An obvious real-world example would be a political election in modern representative democracies. There is a relatively small set of candidates to vote on, but many people who are participating in the elections. From the combinatorial perspective the number of possible linear orders is typically orders of magnitude smaller than the number of voters and thus by the pigeonhole principle some participants must share the same preferences. In such scenarios it may be useful to use weighted elections.

Definition 2.3 (Weighted Election)**.** A pair $E = (C, V, w)$ with $|C| = m$, $|V| = n$ and $w : V \to \mathbb{N}$ is called *a weighted election* if C is nonempty, finite set of candidates, V is a set of preference relations on C, and w is a total function which assigns a weight to each preference.

Note that the formal definition of elections with weighted preferences feels almost like a natural extension of the standard definition. In some contexts an alternative formulation can be used:

Definition 2.4 (Weighted Election (alternative def.))**.** A pair $E = (C, V)$ with $|C| = m$ and $|V| = n$ is called *a weighted election* if C is nonempty, finite set of candidates and V is a multiset of preference relations on C.

To see the benefit of the two above definitions we note that some (not all) algorithms for problems with unique preferences may also work correctly for related problems with weighted preferences using either the first definition, that is by ignoring duplicates, or using the second definition and being unaware of handling duplicates.

Before we can proceed to single-peaked and single-crossing preferences and the algorithms we need more tools (definitions) and notations, most of which we will use extensively throughout this thesis.

Definition 2.5 (Induced election)**.** We call an election $E[C'] = (C', V')$ an induced election of $E = (C, V)$ if $C' \subseteq C$ and $V' = \bigcup_{v \in V}(v \cap (C' \times C'))$, i.e. an election with all candidates but those in C' removed.

The concept of an induced election is extremely useful when dealing with problems involving candidate (or voter) deletion. It allows for succinct, recursive formulation of many problems.

Definition 2.6 (Worst candidate)**.** A *worst candidate* of an election $E = (C, V)$ is a candidate $c_w \in C$ such $\exists v \in V : \forall c_b \in (C \setminus c_w) : (c_w, c_b) \notin v$

A worst candidate is a very intuitive concept: it is simply the candidate ranked last in the election by at least one voter.

Conventions and Notation

When working with elections we always use the letter E to denote an election (regardless whether it contains weighted profiles or not). We use the letters C and V to denote candidate sets and preference sets, respectively. Unless stated otherwise we assume symbols m and n to be always defined as $m = |C|$ and $n = |V|$. We use the subscripted lower case letter v for instance v_1, v_2, v_3 to denote preferences or voters.

For comfort we define two operations typically associated with sets to apply to lists:

Definition 2.7 (List lookup)**.** Let L be a list. We define $\in$ as an operation on L and an element x as follows:

$$x \in L := \begin{cases} \textbf{true} & \textbf{if } L \textbf{ contains } x \\ \textbf{false} & \textbf{otherwise} \end{cases}$$

Definition 2.8 (List concatenation)**.** We define $\cup$ as an operation on lists $L_1 = (x_1, x_2, x_3, \dots, x_k)$, $L_2 = (y_1, y_2, y_3, \dots, y_l)$ as follows:

$$L_1 \cup L_2 := (x_1, x_2, x_3, \dots, x_k, y_1, y_2, y_3, \dots, y_l)$$

Since it can be extremely impractical to input or handle preferences as a set of pairs of candidates (it can be as large as $\leq 2^{\frac{m}{2}}$), there are two common alternative representations used in practice for preference profiles: as lists of candidates and as a matrix (see Fig. 2.1).

$$a \succ_1 b \succ_1 c \qquad \begin{pmatrix} a & c & c \\ b & b & a \\ c & a & b \end{pmatrix}$$
$$c \succ_2 b \succ_2 a$$
$$c \succ_3 a \succ_3 b$$

Fig. 2.1: Two equivalent representations for preferences: as lists (left) and as matrix (right).

The matrix representation can also be transposed (as in Fig. 2.2) so that we get a visual representation very similar to rankings that we have seen in the introductory examples in Section 1.1.

$$\begin{pmatrix} a & b & c \\ c & b & a \\ c & a & b \end{pmatrix}$$

First choice	Second choice	Third choice
a	b	c
c	b	a
c	a	b

Fig. 2.2: Transposed matrix representation of a preference profile (left) and it's ranking (right).

Due to potential ambiguity of the matrix and personal preference we usually employ the list and ranking representations of preference profiles.

2.2 Restricted Domains

After establishing the required terms we can now define two central restricted domains: single-peaked preferences and single-crossing preference profiles.

Single-peaked preference profiles (or single-peaked preferences) are preference profiles with the special property that allows for a linear order over candidates (an axis of candidates) such that for each voter, if we go from left to right we first encounter candidates that are better than all candidates considered before, then we reach the peak candidate, after that each candidate is worse than the one considered before.
For the formal definition of single-peaked preference profiles we follow Bartholdi and Trick [BT86].

Definition 2.9 (Single-peaked preference profile). A preference profile V for a candidate set C is single-peaked if there exists an axis (a list) A containing all candidates from C such that for every preference $v \in V$, A can be split at the most preferred candidate (the peak) of v into two segments (one of which may be empty) X and Y such that for every consecutive a, b in X it holds $a \succ_v b$ and for every consecutive c, d in Y it holds $c \succ_v d$.

Furthermore, for convenience we also define what does it mean for an election to be single-peaked:

Definition 2.10 (Single-peaked election)**.** A (weighted) election $E = (C, V)$ is single-peaked if V is single-peaked.

In the literature, we can find an alternative but equivalent formulation for this problem through the use of forbidden substructures. Forbidden substructure formulations are a mathematical construct allowing to define a structure by absence of substructures with certain properties. Ballester and Haeringer[BH07] describe two types of forbidden substructures defining single-peaked profiles: alpha-configurations and worst-diverse configurations.

Definition 2.11 (Worst-diverse configuration)**.** A profile with three voters v_1, v_2, v_3 and three distinct alternatives a, b, c is a worst-diverse configuration if it satisfies the following:

$$\begin{aligned} v_1 &: \{b, c\} \succ_1 a, \\ v_2 &: \{a, c\} \succ_2 b, \\ v_3 &: \{a, b\} \succ_3 c. \end{aligned}$$

Definition 2.12 (Alpha configuration)**.** A profile with two voters v_1, v_2 and four distinct alternatives a, b, c, d is an α-configuration if it satisfies the following:

$$\begin{aligned} v_1 &: a \succ_1 b \succ_1 c \text{ and } d \succ_1 b \\ v_2 &: c \succ_2 b \succ_2 a \text{ and } d \succ_2 b \end{aligned}$$

Ballester and Haeringer prove the absence of these structures in a profile V as equivalent to this profile being single-peaked [BH07]. Using this equivalence we can alternatively define single-peaked profiles as follows:

Definition 2.13 (Single-peaked preference profile (definition through forbidden substructures))**.** A preference profile V is single-peaked if it contains no worst-diverse configurations and no α-configurations.

Single-crossing preference profiles (or single-crossing preferences) are preference profiles with the special property that they allow for a linear order on voters such that for any pair of candidates $a, b \in C$ the ordering consists of two consecutive segments: one containing all voters with $a \succ b$ and another containing all voters $b \succ c$. An alternative way to think about it is that the axis contains a crossing point for every candidate pair - moving past this point along the axis causes a cross from a group with one preference for the pair to a group with the opposite preference. Formally this can be stated as follows:

Definition 2.14 (Single-crossing preference profile). A preference profile V for a candidate set C is single-crossing if there exists an axis (a list) A containing all preferences of V such that for every pair of candidates $a, b \in C$, A can be split at some point into two consecutive sublists X, Y such that $\forall v \in X : a \succ_v b$ and $\forall v \in Y : b \succ_v a$.

Furthermore, for convenience we also define what does it mean for an election to be single-crossing:

Definition 2.15 (Single-crossing election). A (weighted) election $E = (C, V)$ is single-crossing if V is single-crossing.

Bredereck, Chen and Woeginger[Bre+13a] prove an alternative but equivalent formulation for single-crossing profiles using two types of forbidden substructures: δ-configurations and γ-configurations.

Definition 2.16 (Delta-Configuration). A profile with four voters v_1, v_2, v_3, v_4 and four (not necessarily distinct) candidates a, b, c, d is a δ-configuration if it satisfies the following:

$$\begin{aligned}
v_1 &: a \succ_1 b \text{ and } c \succ_1 d, \\
v_2 &: a \succ_2 b \text{ and } d \succ_2 c, \\
v_3 &: b \succ_3 a \text{ and } c \succ_3 d, \\
v_4 &: b \succ_4 a \text{ and } d \succ_4 d.
\end{aligned}$$

Definition 2.17 (Gamma-Configuration). A profile with three voters v_1, v_2, v_3 and six (not necessarily distinct) candidates a, b, c, d, e, f is a γ-configuration, if it satisfies the following:

$$\begin{aligned}
v_1 &: b \succ_1 a \text{ and } c \succ_1 d \text{ and } e \succ_1 f \\
v_2 &: a \succ_2 b \text{ and } d \succ_2 c \text{ and } e \succ_2 f \\
v_3 &: a \succ_3 b \text{ and } c \succ_3 d \text{ and } f \succ_3 e
\end{aligned}$$

We can thus alternatively define single-crossing preference profiles as follows:

Definition 2.18 (Single-crossing preference profile (definition through forbidden substructures)). A preference profile V is single-crossing if it contains neither δ-configurations nor γ-configurations.

2.3 Nearly Restricted Domains

We can now discuss and define preference profiles which are "almost" single-peaked or "almost" single-crossing.

Consider for example an election $E_x = (C_x, V_x)$ which is not single-peaked but would be if not for a candidate c_x. So if we could remove the candidate c_x from the election E_x (yielding $E_x[C_x \setminus \{c_x\}]$), then the election would become single-peaked. It is then reasonable to say that E_x is nearly single-peaked.
Another example could follow a very similar pattern with an election $E_y = (C_y, V_y)$ which is not single-crossed but would be if not for a voter v_y. So if we could remove the voter v_x from the election E_y (giving us $E_y = (C_y, V_y \setminus \{v_y\})$), then the election would become single-crossing. Again we could reasonably say that E_y is nearly single-crossing.

We could obviously repeat this pattern two more times (deleting a voter for single-peakedness and deleting a candidate for single-crossingness). The question is then how many candidates or voters can we remove from an election and still claim that this election is nearly single-peaked (or single-crossing). Obviously the answer will depend heavily on the concrete application and the context. Therefore, we define a concept of distance for four interesting cases:

Definition 2.19 (Distance functions for restricted domains). A single-peaked candidate distance is a function $d_{\text{SP-CD}} : E = (C, V) \rightarrow \mathbb{N}$ that for any election returns a minimum natural number k such that $\exists C_x \subset C : |C_x| = k \wedge E[C \setminus C_x]$ is single peaked.
A single-peaked maverick distance is a function $d_{\text{SP-MD}} : E = (C, V) \rightarrow \mathbb{N}$ that for any election returns a minimum natural number k such that $\exists V_x \subset V : |V_x| = k \wedge E' = (C, V \setminus V_x)$ is single-peaked.
A single-crossing candidate distance is a function $d_{\text{SP-CD}} : E = (C, V) \rightarrow \mathbb{N}$ that for any election returns a minimum natural number k such that $\exists C_x \subset C : |C_x| = k \wedge E[C \setminus C_x]$ is single-crossing.
A single-crossing maverick distance is a function $d_{\text{SP-MD}} : E = (C, V) \rightarrow \mathbb{N}$ that for any election returns a minimum natural number k such that $\exists V_x \subset V : |V_x| = k \wedge E' = (C, V \setminus V_x)$ is single-crossing.

The above definitions of distance functions can be applied to both weighted and unweighted elections, because all set operations employed in the above descriptions are well-defined for multisets. This feature obviously only comes into play for the two maverick cases.

With regard to the naming convention we decided to stick with the established convention of calling voters which have to be removed "mavericks". The reasoning behind it is that these voters are outliers, hopefully very few who "spoil" an otherwise very structured profile.

From the four distance measures we can construct the four problems SINGLE-PEAKED CANDIDATE DELETION, SINGLE-PEAKED MAVERICK DELETION, SINGLE-CROSSING CANDIDATE DELETION, and SINGLE-CROSSING MAVERICK DELETION as follows:

Definition 2.20 (Problems in Restricted Domains).
Problem: SINGLE-PEAKED CANDIDATE DELETION
Input: An election $E = (C, V)$
Task: Find a subset $C_x \subset C$ such that $d_{\text{SP-CD}} = |C_x|$ and $E[C \setminus C_x]$ is single-peaked.

Problem: SINGLE-PEAKED MAVERICK DELETION
Input: A weighted election $E = (C, V)$
Task: Find a subset $V_x \subset V$ such that $d_{\text{SP-MD}} = |V_x|$ and $E' = (C, V \setminus V_x)$ is single-peaked.

Problem: SINGLE-CROSSING CANDIDATE DELETION
Input: An election $E = (C, V)$
Task: Find a subset $C_x \subset C$ such that $d_{\text{SC-CD}} = |C_x|$ and $E[C \setminus C_x]$ is single-crossing.

Problem: SINGLE-CROSSING MAVERICK DELETION
Input: A weighted election $E = (C, V)$
Task: Find a subset $V_x \subset V$ such that $d_{\text{SC-MD}} = |V_x|$ and $E' = (C, V \setminus V_x)$ is single-crossing.

It should be noted that the SINGLE-PEAKED CANDIDATE DELETION (resp. SINGLE-CROSSING CANDIDATE DELETION) is sometimes called SINGLE-PEAKED ALTERNATIVE DELETION (resp. SINGLE-CROSSING ALTERNATIVE DELETION) in the literature.

For completeness we also define two detection problems for restricted domains: detecting single-crossing and detecting single-peaked profiles. Notice that they are special cases of elections for which $d_{\text{SP-CD}} = d_{\text{SP-MD}} = d_{\text{SC-CD}} = d_{\text{SC-MD}} = 0$, that is the distance is zero[1]

Definition 2.21 (Detecting Restricted Domains).
Problem: SINGLE-PEAKED DETECTION
Input: An election $E = (C, V)$
Task: Compute a witness axis A if E is single-peaked or return **false** otherwise.

[1] A legitimate question would be to ask why bother with the detection algorithms, when the polynomial-time deletion algorithms are practical enough to be used for detection. We discuss this topic in Section 4.3.

Problem: Single-Peaked Detection
Input: A weighted election $E = (C, V)$
Task: Compute a witness axis A if e is single-crossing or return **false** otherwise.

In Fig. 2.3 we present complexity classification of the six above problems, grouped by the type of domain restriction[Erd+13][Bre+13b][Bre+13a][BT86][Elk+12].

Single-Peaked Detection	P
Single-Peaked Candidate Deletion	P
Single-Peaked Maverick Deletion	NP-hard
Single-Crossing Detection	P
Single-Crossing Candidate Deletion	NP-hard
Single-Crossing Maverick Deletion	P

Fig. 2.3: Summarised complexities of six problems, related to single-peakedness and single-crossigness.

2.4 Algorithms for Restricted Domains

After having formally introduced the problems that we are dealing with in this thesis, we provide an overview of the algorithms for them. We group the problems in this section as follows: we first consider the detection algorithms, then we proceed with Single-Peaked Candidate Deletion and Single-Crossing Maverick Deletion, and finish with Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion.

Single-Peaked Detection

Bartholdi and Trick[BT86] first proposed an $\mathcal{O}(m^2n)$-time algorithm for single-peaked detection by reducing the problem to Consecutive Ones Problem and applying an algorithm centred around a data structure called PQ-Tree by Booth and Lueker[BL76]. The $\mathcal{O}(m^2n)$-time algorithm has a nice property that it not only returns one witness axis but *all* of them through the compactness of the PQ-Tree.

Escoffier et al. [Esc+08] proposed a clever alternative algorithm for detecting single-peaked elections, which has linear time complexity ($\mathcal{O}(mn)$). This improvement in the time complexity comes at a cost: the algorithm returns only one axis for single-peaked profiles, whereas the algorithm by Bartholdi and Trick returns a compact representation of all possible axes.. We find especially interesting that the algorithm, which finds a witness axis A, has the same time complexity as an algorithm verifying the witness. Since we are employing this algorithm in our thesis, we will discuss it in some detail.

The general idea is to place candidates, one or two at a time, on an axis until one of two things occurs: there are either no more candidates left to place on the axis — in which case the algorithm successfully found a witness for single-peakedness, or no further candidates can be placed on the axis without violating the single-peakedness constraints.

To ensure the correctness of their algorithm the authors construct the axis outside-in and first place the worst (i.e. the least-preferred) candidates on the opposite ends of the axis. Then they iteratively compute the worst unplaced candidates and place them between the already placed candidates.

The algorithm aborts in one of two cases: it either encounters a set of more than two worst candidates or any placement of the worst candidates on the current axis would violate the single-peakedness constraint. Finding more than two worst candidates at a time is equivalent to finding a worst-diverse configuration. To see this consider that for a worst-diverse configuration three unique voters and three unique candidates are required and each of the voters ranks different candidate as its worst of the three. The checks necessary before placing the worst candidates on the axis eliminate any further possibility of worst-diverse and α-configurations occurring.

Single-Crossing Detection

Elkind, Faliszewski, and Arkadii [Elk+12] first proved that single-crossing detection can be solved in polynomial time. A straightforward implementation of their algorithm has the time complexity $\mathcal{O}(m^2n^2)$.

Bredereck, Chen and Woeginger [Bre+13a] propose two alternative algorithms for single-crossing detection, both with improved time complexity $\mathcal{O}(m^2n)$. One of the algorithms is based on the reduction to the Consecutive Ones problem and an application of the algorithm by Booth and Lueker [BL76]. The other, which we have found to be more interesting and challenging, revolves around the idea of an axis of segments. The algorithm iteratively splits the segments and swaps neighbouring segments until finished or an abort condition is reached.

Since we are using this algorithm in our work we will describe it in more detail. The starting point of the algorithm is an axis with a single segment containing all voters. The order of voters in this segment does not matter.

The algorithm iterates over all (a, b) pairs of candidates, i.e., all possible crossing points to reach a single-crossing axis or an abort condition. During each iteration it goes over the segments on the axis and splits the voters in each segment into two subsegments: one containing voters which prefer a and the other containing voters which prefer b. The two subsegments then replace the original one. If one of the subsegments is empty, then the original segment gets replaced with itself. The ordering of the new subsegments is extremely important: the subsegment placed to the left has to have the same preference regarding (a, b) as its left neighbour (if there is one) and the subsegment placed to the right must have the same preference regarding (a, b) as its right neighbour (if there is one). In case of only one subsegment it has to have the same preference regarding (a, b) as at least one of its neighbours (if there are any). Some special handling is required to properly handle the left-most and right-most segments, but we see it as minor technical detail.

If any of the placements is impossible, then the profile is not single-crossing. Otherwise the algorithm returns a concatenation of the segments from left to right. The crossing point during each iteration is obviously the point where two segments with different preferences regarding a pair meet.

Single-Peaked Candidate Deletion

Erdélyi, Lackner, and Pfandler [Erd+13] proposed a dynamic programming algorithm for SINGLE-PEAKED CANDIDATE DELETION with time complexity $\mathcal{O}(m^5 n)$. The main idea of their algorithm is similar to the single-peaked detection algorithm by Escoffier et al..

Similarly to the single-peaked detection case, the trick in the algorithm is to construct an axis outside-in by placing worst candidates first, at most two candidates at a time, but skipping some candidates. The skipped candidates then constitute the minimum set of candidates to be deleted. To ensure correct computation the authors maintain multiple axes in their state stable throughout the algorithm, and extend them by brute-forcing through worst candidate pairs.

The difficulties we have faced while working with the algorithm and the impracticality of its time complexity has lead us to developing an alternative $\mathcal{O}(m^3 n)$ algorithm for SINGLE-PEAKED CANDIDATE DELETION. We discuss our difficulties and present out algorithm in detail in Chapter 3.

Single-Crossing Maverick Deletion

Bredereck, Chen, and Woeginger proposed[Bre+13b] a $\mathcal{O}(n^4 + n^3 m^2) - time^2$ algorithm for SINGLE-CROSSING MAVERICK DELETION by reducing a weighted election to finding a longest path in a weighted direct acyclic graph (a weighted DAG). Crucial to this reduction is the idea of a disagreement set. A disagreement set of

[2] The time complexity given in the paper is incorrect, since the constructed DAG has $\mathcal{O}(n^4)$ edges and $\mathcal{O}(n^3)$ nodes and not $\mathcal{O}(n^3)$ edges and $\mathcal{O}(n^2)$ as claimed by the authors.

pair of distinct voters v_1, v_2 contains all candidate pairs (a, b) in regard to which the preferences of v_1 and v_2 differ.

Their idea is as follows: it involves generating a DAG with a vertex for each voter pair (plus a starting source vertex) and adding a directed edges between vertices if they are compatible. The compatibility between vertices is then defined as follows: the source vertex is compatible with identity pairs containing one distinct candidate (e.g. $\exists a \in C : (a, a)$). In such case the directed edge goes from the source vertex and has weight 1. Otherwise two vertexes with pairs (v_1, v_2) and (v_3, v_4) are compatible if $v_1 = v_3$ and the disagreement set of (v_1, v_2) is a subset of the disagreement set of (v_3, v_4). The edge goes from (v_1, v_2) to (v_1, v_4) and has the weight equal to the weight of v_4.

Notice that all paths in this DAG start with the source vertex, go through an identity pair, and with each step along the edges the disagreement either stays the same or gets bigger by adding new candidate pairs. Furthermore if we consider the pairs of candidates of the vertexes along all paths we can see that the first candidate is always the same. This guarantees the absence of cycles.

With this the computation of the result is simple: we first find the longest weighted path of the DAG and then collect the second voters from each voter pair along it. This produces a maximum weight set of voters which results in a single-crossing profile. To solve Single-Crossing Maverick Deletion we must then delete all other voters.

Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion

As stated before, Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion are NP-hard and thus a polynomial time algorithm for these problems is impossible (unless P $=$ NP).

We are considering the algorithmic solutions to these problems together, because they are currently very similar.

Elkind and Lackner proposed[EL14] two fixed parameter tractable algorithms (FPT-Algorithms) for Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion which use a reduction through forbidden substructures to a well-known d-Hitting Set problem.

The reduction for Single-Peaked Maverick Deletion is a straightforward one. It involves enumerating all worst-diverse configurations and constructing sets con-

taining voters v_1, v_2, v_3 of a worst-diverse configuration and then enumerating all α-configurations and constructing sets containing voters v_1, v_2 of a α-configuration. This obviously results in an instance of the 3-HITTING SET problem.
The solution to this 3-HITTING SET problem determines the minimum set of voters to delete from the election to get a single-peaked election.
This approach can be generalised to handle weighted elections as well, even though the authors did not mention it.

The reduction for SINGLE-CROSSING CANDIDATE DELETION is very similar to the previous one. It involves enumerating all δ-configurations and constructing sets containing candidates a, b, c, d and then enumerating all γ-configurations and constructing sets containing candidates a, b, c, d, e, f. These configurations contain at most six candidates, which results in an instance of the 6-HITTING SET problem.
The solution to this 6-HITTING SET problem determines the minimum set of candidates to delete from the election to get a single-crossing election.

Through the application of the state-of-the art FPT-Algorithms for the family of d-HITTING SET problems the authors conclude that their algorithms have the time complexity $\mathcal{O}(m^3n^3 + m^4n^2 + 2.08^k)$ for SINGLE-PEAKED MAVERICK DELETION and $\mathcal{O}(m^4n^4 + m^6n^3 + 5.07^k)$ for SINGLE-CROSSING CANDIDATE DELETION.

2.5 Conclusion

Through this brief overview of elections and problems related to single-peaked and single-crossing preferences we have established a set of formal definitions of tools which we will use in later chapters.

In this chapter we have provided an overview of the six problems and algorithms which are implemented in Chapter 4 and allow to perform our analysis of PrefLib in Chapter 5.

Furthermore this chapter frames the context for our SINGLE-PEAKED CANDIDATE DELETION algorithm in Chapter 3.

Theoretical Contribution 3

In this chapter we describe and discuss a polynomial-time candidate deletion algorithm for single-peaked elections. The stated problem for this algorithm is to delete a smallest set of candidates from an election, so that the resulting election is single-peaked. An alternative way to see this problem is to find a largest set of candidates from an election, so that the election restricted to this candidate set is single-peaked.

Our algorithm uses dynamic programming to achieve the goal and, in that respect, is similar to the algorithm by Erdélyi, Lackner, and Pfandler [Erd+13]. Despite key similarities, our algorithm is simpler and more efficient.

3.1 Introduction

Recall from Section 2.4 that Escoffier et al. [Esc+08] proposed a clever linear-time algorithm for detecting and constructing a single-peaked axis for an election if its preference profile is single-peaked. The key idea of the algorithm was that one can start with the left-most and the right-most candidates and construct the axis outside-in. The algorithm starts with the worst candidates (ranked last by at least one voter), and then proceeds to place the candidates ranked second to last to the inside. The candidate placed last on the axis is then a peak candidate (ranked first) for some voter.
An alternative way to see this is to image two rays starting with the left-most candidate and the right-most candidate, facing each other and getting longer with each iteration until they meet, thus completing the axis.

It is nontrivial to apply this algorithm to the Single-Peaked Candidate Deletion problem. Consider for example, that a valid solution can only be found by deleting the candidates that for example ranked last or second to last. (see Fig. 3.1 — placing a and b renders placing c impossible; placing b and c renders placing a impossible). Even if we somehow knew which starting candidates to pick, the next candidate to be placed on the axis is also unknown as the least preferred unplaced candidate may not be a part of any maximum solution (see Fig. 3.2).

To address both of these problems Erdélyi, Lackner, and Pfandler employed [Erd+13] dynamic programming and brute force: instead of knowing or guessing the left-most

$$\begin{aligned} &a \succ_1 b \succ_1 c \\ &c \succ_2 b \succ_2 a \\ &a \succ_3 c \succ_3 b \end{aligned}$$

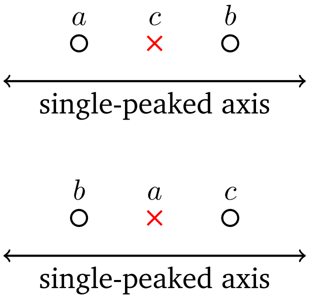

Fig. 3.1: Preference profile for three candidates a, b, c and three voters. Each candidate is ranked last by some voter, hence no two candidates can be designated left-most and right-most without further consideration.

$$\begin{aligned} &a \succ_1 b \succ_1 x \succ_1 c \succ_1 d \\ &a \succ_2 b \succ_2 x \succ_2 d \succ_2 c \\ &b \succ_3 a \succ_3 d \succ_3 x \succ_3 c \end{aligned}$$

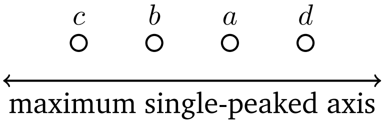

Fig. 3.2: Even after placing two obvious worst candidates, the next worst candidate may not be a part of a maximum solution.

and right-most candidates they consider all pairs as possible left-most and right-most candidates, and instead of placing exactly one or two candidates in each step, they try to place all possible candidate pairs, or every possible axis in each step of the algorithm. This heavy-handed approach comes at great cost to the efficiency: their algorithm has time complexity $\mathcal{O}(m^5 n)$.

Since it seems plausible that there are many applications where elections have relatively few candidates, the time-complexity does not rule out the practicality of the algorithm. Sadly, during the implementation and testing of the algorithm we encountered some issues which ultimately prevented us from completing the task.

The first issue was a minor one: the initialisation step of the algorithm assumes that there is a unique pair of two starting candidates (left-most and right-most candidates on the final axis). This pair is then used to compute partial solutions. It is easy to see that this assumption is false by considering an election with just three candidates as shown in Fig. 3.1. Our solution for this problem was to generate all initial states for all candidate pairs. Obviously (for elections with at least two candidates) there is at least one pair of candidates such that we can safely place one of the left-most and the other right-most on the axis. By considering all pairs, we are certain to come across it. This tweak did neither change the complexity of the algorithm, nor practically slow it down. It is therefore deemed an oversight in the pseudo-code formulation of the algorithm.

$$a \succ_1 b \succ_1 c \succ_1 x \succ_1 y$$
$$c \succ_2 b \succ_2 a \succ_2 x \succ_2 y$$
$$c \succ_3 a \succ_3 b \succ_3 x \succ_3 y$$

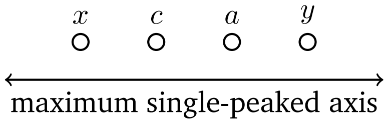

Fig. 3.3: Another problematic case: deciding which two candidates (from three possible ones) to place next, after x and y. The presented solution is one of three maximum solutions.

Two major issues came about when we considered the fine details of implementing the PLACE procedure, i.e., the procedure that decides if the current candidate or the candidate pair can be used to extend some of the partial results already present in the table. Possibly due to space constraints, the authors referred to PLACE procedure by Escoffier et al. [Esc+08] without providing details as to how to apply it correctly. This proved very problematic during the implementation, it is unclear how this procedure should be applied in cases with more than two unplaced least preferred candidates. Consider an example in Fig. 3.3: every maximal solution must contain x and y as the outermost candidates, but at least one of a, b, or c must be deleted. Invoking the PLACE procedure three times with $\{c\}$ first, followed by $\{b\}$, and followed by $\{a\}$ would still result in an axis, albeit not a single-peaked one.

As for the second issue: the presented proof is very short and for our needs, it lacks depth at several steps (thus enabling and magnifying the issues described above). As far as we know there is no longer version of the proof[1]. At this point we came to the conclusion that the issues proved too much and we had to abandon the algorithm. Nevertheless the intensive work done on the algorithm allowed us to create an alternative, improved algorithm which while employing the same core ideas, has lower time complexity and can be easily implemented.

Our dynamic programming algorithm starts with every candidate pair as an initial state in the state table. Each entry in the table is identified primarily by two candidates which are the current "center" of an axis and also the boundary between which further candidates can be placed. The algorithm then iterates over all candidates ordering them by how "bad" are they: first come candidates ranked last by some voter, then candidates ranked second to last by some voter and so on. During each iteration our algorithm tries to place the currently considered candidate between every possible pair. If a placement is valid (it does not violate single-peaked axis constraints), we compare a potential new entry (an old entry with new candidates added between the boundaries) with an existing entry in the table with the same candidates closest to the middle. If the potential entry contains more candidates than an existing one, then we store it in the state table instead of the entry we have

[1] We ware able to obtain an extended, pre-print version of the relevant research from arXiv.org (`http://arxiv.org/abs/1211.2627`). Sadly this version did not expand upon the proof or the algorithm's details.

$$d \succ_1 a \succ_1 b \succ_1 c \succ_1 x \succ_1 y$$
$$d \succ_2 c \succ_2 b \succ_2 a \succ_2 x \succ_2 y$$
$$d \succ_3 c \succ_3 a \succ_3 b \succ_3 x \succ_3 y$$

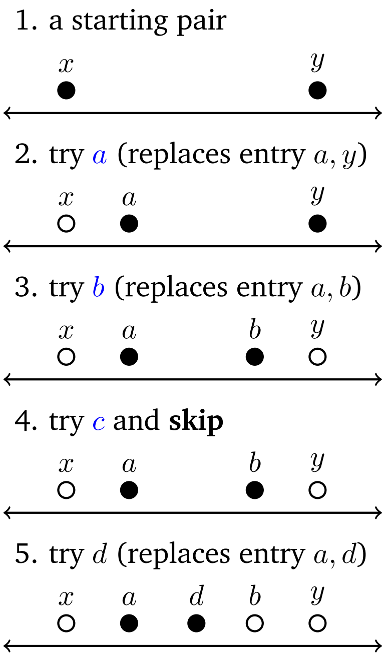
1. a starting pair
x
y
2. try a (replaces entry a, y)
x
a
y
3. try b (replaces entry a, b)
x
a
b
y
4. try c and skip
x
a
b
y
5. try d (replaces entry a, d)
x
a
d
b
y

Fig. 3.4: Partial trace of the algorithm's execution which results in a maximum solution by skipping over c from the six candidates. Note that at most two of the a, b, c candidates can be placed on the axis in a valid solution.

compared it to, otherwise we discard it. Ties do not matter in our algorithm, because we only search for one maximum solution and not all of them. The final step in our algorithm is to find a an entry in the table with largest number of candidates and return them. Notice that there is no explicit delete operation in this algorithm. Deletion of a candidate c happens when we replace an axis containing c with some other axis without c in the state table, and this new axis propagates through computation up until the final solution. It may be more appropriate to think of skipping the candidates instead of deleting them when considering this algorithm as presented in Fig. 3.4.

From the above description it may not be immediately clear, why this algorithm should produce a global maximum. There are two ways to intuitively understand the algorithm considering the output axis: inside-out and outside-in. We first describe the inside-out view. Consider a global maximum that the algorithm has found. Now there are (at least) two candidates which are ranked higher by at least one voter than any others. We consider this pair of candidates to be the center of an axis stored inside the state table. If we trace the execution of the algorithm one iteration back, we will find another almost identical entry in the state table, except one of the candidates from the center is missing, and exactly this candidate is encountered by the algorithm and considered for placement on all axes present in the state table. Continuing this decomposition we arrive at the pair of the left-most and the right-most candidate — one of the initial states in the table.

The outside-in view is the opposite: there is a starting pair which will lead to a global maximum by placing candidates between them. During each iteration the algorithm will finally encounter the candidate that should be placed if there is such candidate, and this axis will propagate through the iterations until the final solution is computed.

The two key differences in our algorithm to the one which inspired it are that during each iteration of the inner loop we place (at most) two candidates but only one at a time instead of trying all pairs and we cleverly compute the list of candidates considered for placement. In this way we were able to reduce the time complexity from $\mathcal{O}(m^5 n)$ to $\mathcal{O}(m^3 n)$ (each difference reduces the complexity by $\mathcal{O}(m)$).

3.2 Single-Peaked Candidate Deletion

Before we can introduce the dynamic programming algorithm for single-peaked candidate deletion, we need to formally defined the concepts introduced in the previous section. In this chapter we also the use the definitions from the Chapter 2.

Definition 3.1 (single-peaked axis)**.** An ordered list or an *axis* A is single-peaked with respect to a preference relation v_v if A orders the candidates such that consecutive candidates until a candidate c_p (including c_p) are preferred to all candidates before them, and consecutive candidates after c_p (including c_p) are preferred to all candidates after them.
A is single-peaked with respect to an election $E = (C, V = (v_1, \dots, v_m))$ if A is single-peaked with respect to every v_i.

The following definition of a pp-axis (a partitioned partial axis) is very similar to the definition of a partial axis from the paper by Escoffier et al. [Esc+08].

Definition 3.2 (pp-axis)**.** A pp-axis P for an election $E = (C, V)$ with $m = |C|$ and $n = |V|$ is a partition of the candidate set C into two disjoint, non-empty, and ordered candidate lists (C_1, C_2), such that for at least one voter $v \in V$ the candidates in C_1 are ordered in increasing preference and the candidates in C_2 are ordered in decreasing preference for this voter, and the concatenation of $C1$ and $C2$ results in an axis A which is single-peaked with respect to E.
The tuple of the last candidate in C_1 and the first candidate in C_2 is the **boundary** of P.

Definition 3.3 (Axis configurations)**.** A pp-axis $P = (C_1, C_2)$ is **normal**, when both C_1 and C_2 contain at least two candidates each. A pp-axis P is **left-skewed** when C_1 contains only one candidate. A pp-axis P is **right-skewed** when C_2 contains only one candidate.
Normal, **left-skewed**, **right-skewed** are (axis) configurations.

The significance of the boundary is twofold: it not only limits the lists C_1 and C_2 but also provides place where new candidates can be added — between the boundary. Consider as wall that the boundary slowly converges to a peak candidate throughout the algorithm.

Note that the axis configurations are a purely artificial construct which we employ to more easily prove the correctness of the algorithm. A practical implementation of the algorithm would probably do without special handling of configurations (as did ours). For examples of axis configurations see Fig. 3.5.

$$\begin{array}{ll}
c \succ_1 d \succ_1 b \succ_1 a & (a, b), (c, d) \\
a \succ_2 b \succ_2 c \succ_2 d & (a), (b, c, d) \\
d \succ_3 c \succ_3 b \succ_3 a & (a, b, c), (d)
\end{array}$$

Fig. 3.5: An election with four candidates and three voters followed by three example partitionings that can be constructed from it: **normal**, **left-skewed** and **right-skewed**.

The definitions enable a couple of observations which are useful to better understand our algorithm.

Observation 3.1. Further properties of pp-axes:

(1) Every single-peaked axis can be partitioned into at least one pair of suborders C_1 and C_2 such that (C_1, C_2) is a pp-axis.

(2) Any partitioning of a set with two distinct candidates into non-empty ordered lists is pp-axis and thus induces a single-peaked election.

(3) For every pp-axis $P = (C_1, C_2)$ there exists another pp-axis $P_s = (\text{reverse}(C_2), \text{reverse}(C_1))$. We thus call every pp-axis symmetric.

(4) For any given single-peaked axis there can be multiple pp-axes.

We use the following conventions and notation when working with pp-axes:

- For a pp-axis $P = (C_1, C_2)$ with:

$$C_1 = (l_{|C_1|}, l_{|C_1|-1}, l_{|C_1|-2}, l_{|C_1|-3}, \ldots, l_1)$$
$$C_2 = (r_1, r_2, r_3, \ldots, r_{|C_2|})$$

  we use the notation:

$$(l_{|C_1|}, l_{|C_1|-1}, l_{|C_1|-2}, l_{|C_1|-3}, \ldots, l_1, \leftrightarrow r_1, r_2, r_3, \ldots, r_{|C_2|})$$

- For brevity we use $(\ldots l \leftrightarrow r \ldots)$ to argue about a set of pp-axes with candidate lists C_1, C_2, such that l is the last candidate of C_1 and r is the first candidate of C_2. The dots symbolize that C_1 (respectively C_2) may contain other candidates than l (respectively r).

- When referring to multiple candidates on the axis we use indexes: l_1 and r_1 for the boundary, l_2 is the candidate immediately to the left of l_1 and r_2 is the candidate immediately to the right of r_2, and so on. To ease understanding we abbreviate the notation of a pp-axis with such configuration to:

$$(\ldots l_3 l_2 l_1 \leftrightarrow r_1 r_2 r_3 \ldots).$$

- When referring to axis configurations, for brevity we may use symbols $\mathtt{N}, \mathtt{L}, \mathtt{R}$ for **normal**, **left-skewed**, **right-skewed** respectively. We use the symbol z to denote a configuration.

- In situations where it is clear from the context we may refer to a pp-axis simply as axis.

Definition 3.4 (Induced election (by a pp-axis)). Let $P = (C_1, C_2)$ be a pp-axis. We call an election $E[P] = (C_1 \cup C_2, V')$ an induced election of $E = (C, V)$ if $E[P] = E[C_1 \cup C_@]$.

Recall from Section 2.1 that the notion of induced election is especially useful when reasoning about problems which involve deleting candidates from an election, such as deleting a minimum set of candidates so that the resulting election is single-peaked. We extend this notion to pp-axes. Furthermore we use induced elections to construct a partially ordered list of candidates "from worst to best". This list we call worst candidate list.

Definition 3.5 (Worst candidate list). A *worst candidate set* W of E is then a set of candidates $W \subseteq C$, such that every $c \in W$ is a worst candidate of E.
A *worst candidate list* $\mathcal{W}$ of $E = (C, V)$ is a list of worst candidate sets $\mathcal{W} = (W_1, W_2, W_3, \ldots, W_k)$, such that:

- W_1 is the worst candidate set of E,

- W_i with $1 < i \leq k$ is the worst candidate set of $E[C \setminus \bigcup_{j=1}^{i-1} W_j]$, and

- $\bigcup_{j=1}^{k} W_j = C$.

The construction of the worst candidate list of E can also be described iteratively: in the initialisation step find the worst candidate set and append it to $\mathcal{W}$. Execute the following until the the election is empty: let W_i be the last worst candidate set appended to $\mathcal{W}$. Replace E with an induced election $E[C \setminus W_i]$. Find the worst candidates set of the election and append it to $\mathcal{W}$. Increment i. It is obvious from this formulation that the worst candidate sets are disjoint.

Definition 3.6 (Maximum pp-axis)**.** We call an integer i a *level* if $W_i \in \mathcal{W}$ exists. Furthermore we call candidates in W_i candidates from *level* i and define $\forall c \in W_i$: $level(c) = i$.
We call an integer i_{max} *last level* if there exists a $W_{i_{max}}$ and it is the last set in $\mathcal{W}$. In the degenerate case of an election with no candidates, we arbitrarily define i_{max} to zero.
A pp-axis P for (not necessarily all) candidates from $\bigcup_{k \leq i} W_k \cup \{l, r\}$ is a *maximum pp-axis* with boundary l, r, configuration z at level i, when the following conditions are fulfilled:

- l, r ist the boundary of P,
- besides l, r, axis P contains only candidates from levels up to i,
- if z is left-skewed (resp. right-skewed) then P is left-skewed (resp. right-skewed),
- if $level(l) > i$ then P is left-skewed,
- if $level(r) > i$ then P is right-skewed, and
- no other axis with more candidates exists which fullfills all the above criteria.

The intuition for this definition of maximality is as follows: delete all candidates from all levels higher than i and construct the maximum axis. For the cases when the axis would be empty, or the axis would contain only one candidate we allow the axis to contain l and r. We are allowed to do this because of Observation 3.1.(2).

We now introduce the crucial part of our dynamic programming algorithm for single-peaked candidate deletion: the pp-axis table.

Definition 3.7 (pp-axis Table.)**.** A table $\mathcal{T}(l, r, i, z)$ for an election $E = (C, V)$ is called a (maximum) pp-axis table if for every $l, r \in C, 0 \leq i \leq i_{max}, z \in \{\mathtt{N}, \mathtt{L}, \mathtt{R}\}$ it contains a maximum pp-axis with boundary l, r, configuration z at level i.

The table $\mathcal{T}$ has few crucial properties which we use in our algorithm. To fully exploit them we first introduce and prove the following two lemmas.

Lemma 3.2. *Let $P(C_1, C_2)$ be a pp-axis. There are no two candidates c_i, c_j with $level(c_i) > level(c_j)$ such that either*

- *c_i comes ahead of c_j in C_1, or*
- *c_i comes behind c_j in C_2.*

Proof of Lemma 3.2. We show the first part by contradiction. Let there be a pp-axis $P(C_1, C_2)$ with two candidates c_i, c_j such that $level(c_i) > level(c_j)$.
Suppose for the sake of contradiction that c_i comes ahead of c_j in C_1. By the definition of $level(*)$ for all preferences $v \in V$ it holds that c_i is preferred to c_j by v. But this contradicts the definition of a pp-axis: that there must exist a voter $v_s \in V$, so that the candidates in C_1 are ordered in an increasing preference of v_s. Thus we arrive at a contradiction and prove the initial claim.
For the second part we use the same scheme and argue symmetrically for the decreasing order in C_2 for some voter $v_s \in V$. □

Lemma 3.3. *Let $P(C_1, C_2)$ be a pp-axis. For all $W_i \in \mathcal{W}$ it holds that $|W_i \cap (C_1 \cup C_2)| \leq$ 2.*

Proof of Lemma 3.3. By Lemma 3.2 we can infer that candidates from the same level must either occupy consecutive positions in one of the lists of the pp-axis or be split between the two lists. We will show that placing three (or more) candidates from one level leads to a situation which is impossible to resolve. Let c_1, c_2, c_3 be three candidates from the same level that we want to place on a pp-axis. W.l.o.g. assume that c_1 and c_2 are a boundary of some pp-axis (see Observation 3.1.(2)). We now try to place c_3. Consider that for each of the candidates c_1, c_2, c_3, there is a voter $v_i \in V$ that ranked c_i last. Now consider any placement of c_3, and the candidate c_i which ends in the middle, that is between the two other candidates on the axis. Then there is a voter $v_i \in V$ which ranked c_i last, leading to a violation of single-peakedness property of the axis: for this axis and the voter v_i the candidates are ordered in decreasing preference, followed by increasing preference.
This problem occurs regardless how many candidates we place before, after, or between candidates c_1, c_2, c_3, therefore we can conclude that it is impossible to place more than two candidates from any level. □

Lemma 3.2 and Lemma 3.3 can be summarised succinctly as follows: for two candidates c_1, c_2 from one partition with $level(c_1) > level(c_2)$, c_1 must be closer to the boundary than c_2, and any pp-axis can contain at most two candidates from a given level.

Lemma 3.3 represents the core insight from the $\mathcal{O}(mn)$ single-peaked axis construction algorithm by Escoffier et al.[Esc+08]. Together with Lemma 3.2 it allows for an algorithm which iterates over worst candidate list and places the candidates outside-in.

The consequence of the Lemma 3.3 is very interesting: finding maximum candidate set which induces a single-peaked election is equivalent to removing surplus candidates (i.e. some candidates from worst candidate sets with size larger two) and removing any other conflicting candidates. Obviously there are potentially exponentially many candidate sets which can be removed rendering a brute-force algorithm infeasible.

Now we consider the properties of the pp-axis table $\mathcal{T}$ which are of interest from the point of view of our algorithm. The key insight here is that $\mathcal{T}$ can be constructed through dynamic programming, one level at a time. We also observe the following:

Observation 3.4. For every $x, y \in C, z \in \{\mathtt{N}, \mathtt{L}, \mathtt{R}\}$ it holds, that

$$\mathcal{T}(l, r, i > max(level(l), level(r)), z) = \mathcal{T}(l, r, max(level(l), level(r)), z).$$

This directly follows from Lemma 3.2. Once the last worst set containing either l or r has been reached, no more candidates can be added – adding them between l and r would lead to a different boundary and thus a different entry in $\mathcal{T}$, and adding them somewhere else would violate Lemma 3.2.
Alternatively one can think of it as follows: for l and r to be the boundary of a pp-axis, one must delete every candidate from the election which are in levels above l and r.

Lemma 3.5. *Let $E = (C, V)$ with $m = |C|$ and $n = |V|$ be an election and $\mathcal{T}$ be the maximum pp-axis table for this election, then $\mathcal{T}$ has size $\mathcal{O}(m^4)$ and contains a maximum pp-axis $A = (C_1, C_2)$, such that $E[C_1 \cup C_2]$ is a maximum single-peaked election.*

Proof of Lemma 3.5. We prove the claim that $\mathcal{T}$ contains a maximum pp-axis by contradiction. Let $E = (C, V)$ be an election, $\mathcal{T}$ be the maximum pp-axis table for this election, P be a maximum pp-axis for this election, and i_{max} be the maximum level. By our assumption, P must not occur in $\mathcal{T}$ and must be longer than any pp-axis in $\mathcal{T}$. Now consider the boundary l_P, r_P and the configuration z_P of P. By our assumption $\mathcal{T}(l_P, r_P, i_{max}, z_P)$ must contain a shorter pp-axis than P which contradicts the definition of $\mathcal{T}$.
We have thus proven that $\mathcal{T}$ contains a maximum pp-axis for election E. It is trivial to see that this axis induces a globally maximal single-peaked election.

We now consider the size of the table $\mathcal{T}$. Let $E = (C, V)$ be an election with $m = |C|$ and $n = |V|$ and $\mathcal{T}$ be a pp-axis table that has been computed for E. The number of

entries in $\mathcal{T}$ is then upper-bounded by $m \cdot (m-1) \cdot (m+1) \cdot 3$. To see this, consider that they are m possible x-values, $m-1$ possible y-values, at most m sets in the worst candidate list (each set with exactly one candidate), and therefore at most $m+1$ levels, and exactly 3 possible configurations for each boundary x, y.
Thus we can conclude that $\mathcal{T}$ has $\mathcal{O}(m^3)$ entries, and each entry is trivially upper-bounded by $\mathcal{O}(m)$ which results in a total space complexity of $\mathcal{O}(m^4)$. □

By Lemma 3.5 we get that computing $\mathcal{T}$ also gives an induced election that is single-peaked and maximum, i.e. there is no other larger set of candidates that also induces a single-peaked election. Solving the actual problem is then trivial: we substract the computed maximum from the candidate set. The result contains the candidates that have to be deleted to obtain a single-peaked election. Notice that we only get a maximum (resp. minimum), and we are not guaranteed to find all maxima (resp. all minima).

Algorithm 1 single-peaked candidate deletion algorithm

```
global: $\mathcal{T}$ is an empty pp-axis table
procedure SPCandidateDelete($E = (C, V)$)
    for $l, r, z \in (C \times C \times \{\mathtt{N}, \mathtt{L}, \mathtt{R}\})$ do
        $\mathcal{T}(l, r, 0, z) \leftarrow (l \leftrightarrow r)$
    $\mathcal{W} \leftarrow$ WorstCandidateList($E$)
    for $W_i \in \mathcal{W}$ do
        for $l, r, z \in ((C \times C \setminus \{(a, a) : a \in C\}) \times \{\mathtt{N}, \mathtt{L}, \mathtt{R}\})$ do
            $\mathcal{T}(l, r, i, z) \leftarrow \mathcal{T}(l, r, i-1, z)$
        for $c_1 \in W_i$ do
            for $l, r, z \in ((C \times C \setminus \{(a, a) : a \in C\}) \times \{\mathtt{N}, \mathtt{L}, \mathtt{R}\})$ do
                Place($E, l, r, i, z, c_1$)
        for $c_2 \in W_i$ do
            for $l, r, z \in ((C \times C \setminus \{(a, a) : a \in C\}) \times \{\mathtt{N}, \mathtt{L}, \mathtt{R}\})$ do
                Place($E, l, r, i, z, c_1$)
    return Max($\mathcal{T}$)
```

Algorithm 2 Worst candidate list algorithm

```
procedure WorstCandidateList($E = (C, V)$)
    if $C = \emptyset$ then
        return *Empty List*
    $W \leftarrow \{\}$
    for $v \in V$ do
        $c \leftarrow$ worst candidate of $v$
        $W \leftarrow W \cup \{c\}$
    return $(W) \cup$ WorstCandidateList($E[C \setminus W]$)
```

Lemma 3.6. *The* Place *procedure in Algorithm 3 is correct, that is, it only generates valid pp-axes and only replaces entries in the table $\mathcal{T}$ if the new entry contains more candidates.*

Algorithm 3 Place procedure

```
global: $\mathcal{T}$ is an empty maximum pp-axis table with entries at level $i$
procedure PLACE($E = (C, V), l, r, i, z, x$)
    $(\ldots l \leftrightarrow r \ldots) \leftarrow \mathcal{T}(l, r, i, z)$
    if $z = \texttt{L}$ then
        if $(\ldots l \leftrightarrow xr \ldots)$ is single-peaked and left-skewed then
            SETIFLONGER($l, x, (\ldots l \leftrightarrow xr \ldots), i, z$)
        if $(\ldots lx \leftrightarrow r \ldots)$ is single-peaked then
            SETIFLONGER($x, r, (\ldots lx \leftrightarrow r \ldots), i, \texttt{N}$)
    if $z = \texttt{R}$ then
        if $(\ldots lx \leftrightarrow r \ldots)$ is single-peaked and right-skewed then
            SETIFLONGER($x, r, (\ldots lx \leftrightarrow r \ldots), i, z$)
        if $(\ldots l \leftrightarrow xr \ldots)$ is single-peaked then
            SETIFLONGER($l, x, (\ldots l \leftrightarrow xr \ldots), i, \texttt{N}$)
    if $z = \texttt{N}$ then
        if $(\ldots lx \leftrightarrow r \ldots)$ is single-peaked then
            SETIFLONGER($x, r, (\ldots lx \leftrightarrow r \ldots), i, z$)
        if $(\ldots l \leftrightarrow xr \ldots)$ is single-peaked then
            SETIFLONGER($l, x, (\ldots l \leftrightarrow xr \ldots), i, z$)

procedure SETIFLONGER($l, r, P_{new}, i, z$)
    if $|P_{new}| > |\mathcal{T}(l, r, i, z)|$ then
        $\mathcal{T}(l, r, i, z) \leftarrow P_{new}$
```

Proof of Lemma 3.6. The correctness of the PLACE procedure is very easy to deduce from the construction of the algorithm. To see that it only generates valid pp-axes, consider that for each of the cases $z \in \{\texttt{L}, \texttt{N}, \texttt{R}\}$ it checks the whether a potential new pp-axis conforms to the constraints of the given configuration and only then calls the SETIFLONGER procedure. It is trivial to see that the SETIFLONGER procedure replaces shorter axes with longer axes. □

Algorithm 1 contains a bottom-up formulation of the algorithm as pseudo-code. To ease the understanding of the algorithm and of the proof we also offer an alternative top-down formulation in Fig. 3.6, more typical for dynamic programming algorithms. To ensure that both variants always produce the same result (especially in cases with multiple maxima) we define max function to always return the longest axis from the table $\mathcal{T}$. In case of ties it resolves them the same way as they are resolved in the bottom-up variant. This can be ensured by enforcing a special order (for example lexicographic order over level, candidates, and configurations) on the entries in the table $\mathcal{T}$. For brevity let lv be an alias for $level$ and "is sp" mean "is single-peaked".

Theorem 3.7. *Algorithm 1 computes $\mathcal{T}$ and returns a maximum set of candidates that induces a single-peaked election in $\mathcal{O}(m^3 n)$ time.*

$$
\mathcal{T}(l_1, r_1, i, z) = \begin{cases}
(l_1 \leftrightarrow r_1) & \textbf{if } (lv(l_1) \neq i) \wedge (lv(r_1) \neq i) \\
(l_1 \leftrightarrow r_1 r_2 r_3 \dots) & \textbf{if } (lv(l_1) \neq i) \wedge (lv(r_1) = i) \wedge \\
& \exists r_2, r_3 \in C \text{ with:} \\
& (l_1 \leftrightarrow r_1 r_2 r_3 \dots) \text{ is sp } \wedge (lv(r_2) = i) \wedge \\
& (l_1 \leftrightarrow r_3 \dots) = max_{c \in C}(\mathcal{T}(l_1, c, i-1, \mathtt{L})) \\
(l_1 \leftrightarrow r_1 r_2 \dots) & \textbf{if } (lv(l_1) \neq i) \wedge (lv(r_1) = i) \wedge \\
& \exists r_2 \in C \text{ with:} \\
& (l_1 \leftrightarrow r_1 r_2 \dots) \text{ is sp } \wedge \\
& (l_1 \leftrightarrow r_2 \dots) = max_{c \in C}(\mathcal{T}(l_1, c, i-1, \mathtt{L})) \\
(\dots l_3 l_2 l_1 \leftrightarrow r_1) & \textbf{if } (lv(l_1) = i) \wedge (lv(r_1) \neq i) \wedge \\
& \exists l_2, l_3 \in C \text{ with:} \\
& (\dots l_3 l_2 l_1 \leftrightarrow r_1) \text{ is sp } \wedge (lv(l_2) = i) \wedge \\
& (\dots l_3 \leftrightarrow r_1) = max_{c \in C}(\mathcal{T}(c, r_1, i-1, \mathtt{R})) \\
(\dots l_2 l_1 \leftrightarrow r_1) & \textbf{if } (lv(l_1) = i) \wedge (lv(r_1) \neq i) \wedge \\
& \exists l_2 \in C \text{ with:} \\
& (\dots l_2 l_1 \leftrightarrow r_1) \text{ is sp } \wedge \\
& (\dots l_2 \leftrightarrow r_1 \dots) = max_{c \in C}(\mathcal{T}(c, r_1, i-1, \mathtt{R})) \\
(\dots l_2 l_1 \leftrightarrow r_1 r_2 \dots) & \textbf{if } (lv(l_i) = i) \wedge (lv(r_1) = i) \wedge \\
& \exists l_2, r_2 \in C \text{ with:} \\
& (\dots l_2 l_1 \leftrightarrow r_1 r_2 \dots) \text{ is sp } \wedge \\
& (\dots l_2 \leftrightarrow r_2 \dots) = max_{c, d \in C, z' \in \{\mathtt{L},\mathtt{R},\mathtt{N}\}}(\mathcal{T}(c, d, i-1, z')) \\
\mathcal{T}(l_1, r_1, i-1, z) & \textbf{otherwise}
\end{cases}
$$

Fig. 3.6: Top-down computation of the state table $\mathcal{T}$ in the Single-Peaked Candidate Deletion algorithm

Proof of Theorem 3.7. We prove that the algorithm constructs $\mathcal{T}$ by induction over the levels in the worst candidates list.

Base case: $i = 0$.
By construction of the algorithm, the first step initializes the table at level 0, so that each entry contains the respective boundary as the maximum pp-axis. By definition of $\mathcal{T}$ all entries at level 0 are maximum pp-axes.

Induction hypothesis: All entries in $\mathcal{T}$ for the level i are maximum pp-axes.

Induction step: It remains to show that after executing the algorithm all entries in $\mathcal{T}$ for the level $i+1$ are maximum pp-axes. We actually only show that after $i+1$st iteration of the outer loop of the algorithm, the table contains maximum pp-axes for the level $i+1$. We can do this because the algorithm does not change entries from lower levels after processing them — as a consequence of the Lemma 3.2. Assume, from now on, that the algorithm has already executed $i+1$th iteration of the outer loop.

First we consider that an entry of $\mathcal{T}$ from level $i+1$ can be any of the three configurations: $\{\mathtt{L}, \mathtt{N}, \mathtt{R}\}$. Therefore it is sufficient to show the correctness for each of the configurations to prove the induction step. The first part proves *the normal case*, that is an entry with configuration N, the second part *the left-skewed case*, that is an entry with configuration L, and the third *the right-skewed case*, that is an entry with configuration R.

Part 1. The normal case:

Let $(\ldots l_1 \leftrightarrow r_1 \ldots)$ be an entry in $\mathcal{T}$ for level $i+1$ with configuration N. Now consider the two candidates: l_2 which is to the left of l_1, and r_2 which is to the right of r_2. If either l_2 or r_2 does not exist, then this axis is actually **left-skewed** or **right-skewed** axis (despite being labeled as **normal**), and the appropriate case applies. Note that this includes the special case, where both l_1 and r_1 are from levels higher than $i+1$. In this special case the axis contains only l_1 and r_1 and is by definition of a maximum pp-axis for this level.
We will now show that $(\ldots l_2 l_1 \leftrightarrow r_1 r_2 \ldots)$ in $\mathcal{T}$ for level $i+1$ with configuration N is a maximum pp-axis.
We distinguish between several sub-casest:

1. l_1 is from a level higher than $i+1$ and r_1 is from a level lower than $i+1$ (or the situation is reversed),

2. l_1 is from a level higher than $i+1$ and r_1 is from the level $i+1$ (or the situation is reversed),

3. l_1 and r_1 are from the level $i+1$,

4. l_1 is from level $i+1$ and r_1 is from a level lower than $i+1$ (or the situation is reversed).

Due to Observation 3.1.(3) we can assume without loss of generality that a proof for one sub-case also holds when the situation is reversed (symmetrical).

For the first two sub-cases note that they cannot occur: since l_1 is from a level higher than $i+1$, it has not been yet processed, and there is no candidate l_2. Therefore the argumentation for the **left-skewed** case (or **right-skewed** configuration if situation in reversed) applies.

The third case is the most straightforward. By Lemma 3.2 candidates l_2 and r_2 are from levels lower than l_1 and r_1. By the induction hypothesis we can assume that $\mathcal{T}(l_2, r_1, i, \mathtt{N})$ contains a maximum pp-axis. Note that the algorithm considers all the entries in $\mathcal{T}$ from level i and tries to place every candidate from the level $i+1$ and then every candidate from the same level, therefore we can conclude that at some point the PLACE procedure has been called on $(\ldots l_2 \leftrightarrow r_2 \ldots)$ with l_1, and then again on $(\ldots l_2 l_1 \leftrightarrow r_2 \ldots)$ with r_1, finally generating $(\ldots l_2 l_1 \leftrightarrow r_1 r_2 \ldots)$, if such axis is possible. By Lemma 3.3 we can conclude that $(\ldots l_2 l_1 \leftrightarrow r_1 r_2 \ldots)$ is a maximum pp-axis with boundary l_1, r_1 and configuration $\mathtt{N}$ for the level $i+1$.

For the fourth sub-case consider that no candidates can be placed to the the right of l_1 because of Lemma 3.2. Therefore for the maximality $\mathcal{T}(l_1, r_1, i+1, \mathtt{N})$, we must only consider the candidates l_2 (and possibly l_3) that could have been placed to the right of l_1 This leads to another two sub-cases:

1. r_2 is from a level lower than $i+1$ or

2. l_2 is from level $i+1$.

If l_2 is from a level lower than $i+1$, then we can conclude that no other candidate from $i+1$ can be placed between l_2 and l_1 — because by its construction the algorithm (also) tries to place every other candidate from level $i+1$ before placing l_1, and since it failed, no such axis can exist. Through the induction hypothesis, i.e. $\mathcal{T}(l_2, r_1, i, \mathtt{N})$ being maximum, we can conclude that $\mathcal{T}(l_1, r_1, i+1, \mathtt{N})$ is maximum.

If l_2 is from the same level as l_1, then by Lemma 3.3 we know that no other candidates from level $i+1$ could have been placed. We thus need to consider a possible candidate l_3. A similar reasoning to the third case applies: by the induction hypothesis we know that $\mathcal{T}(l_3, r_1, i, \mathtt{N})$ is maximum, and that the algorithm successfully has tried to place l_2 and after that l_1 to the right of l_3. Therefore, we can conclude that $(\dots l_3 l_2 l_1 \leftrightarrow r_1 \dots)$ is maximum for level $i+1$ and has been stored in $\mathcal{T}(l_1, r_1, i+1, \mathtt{N})$. On the other hand, if there is no candidate l_3, then the left half of $\mathcal{T}(l_2, r_1, i, \mathtt{N})$ which by induction hypothesis is maximum, contains only l_2. By Lemma 3.3 only one candidate could have been added, and if $(\dots l_2 l_1 \leftrightarrow r_1 \dots)$ is a valid pp-axis, then the algorithm would have considered it and stored it in $\mathcal{T}(l_1, r_1, i+1, \mathtt{N})$.

Notably l_3 cannot be from a higher level than $i+1$ because of Lemma 3.2.

This argumentation obviously also holds true if the situation is reversed, that is r_1 is from level $i+1$ and l_1 is from level lower than $i+1$.

Part 2. The left-skewed case.

Let $(\dots l \leftrightarrow r_1 \dots)$ be an entry in $\mathcal{T}$ for level $i+1$ with configuration $\mathtt{L}$. Now consider candidate r_2 which is to the right of r_1. If r_2 does not exist then l and r_1 are from levels higher than $i+1$ and the entry in $\mathcal{T}$ is by definition maximum.

Because of the requirements of the configuration $\mathtt{L}$, the left half of all entries in $\mathcal{T}$ with the left boundary l and configuration $\mathtt{L}$ will only contain l. Therefore we can safely ignore it in our analysis and focus on the right half. For this we consider the levels of r_1 and r_2 as sub-cases.

- r_1 and r_2 are from the level $i+1$.
- r_1 is from level $i+1$ and r_2 is from the a level lower than $i+1$

The following argumentation is intentionally kept short, because the cases are almost identical to the ones already discussed. Thus some assumptions and lemmas are implicit.

If both r_1 and r_2 are from the level $i+1$ we consider the existence of candidate r_3 to the right of r_2. If it exists, then is must be from a lower level than r_1. By induction hypothesis $\mathcal{T}(l, r_3, i, \mathtt{L})$ contains a maximum pp-axis $(l \leftrightarrow r_3 \dots)$. Similarly to the argumentation above, we can conclude that, since the algorithm considers all ordered pairs from the level $i+1$ and since r_1 and r_2 are a valid pair which can be placed on the pp-axis $(l \leftrightarrow r_3 \dots)$, $\mathcal{T}(l, r_1, i+1, \mathtt{L})$ indeed contains a maximum pp-axis, namely the extension of $(l \leftrightarrow r_3 \dots)$. If, on the other hand, r_3 does not exist, then only one candidate could have been placed during this iteration on the axis $(l \leftrightarrow r_2 \dots) = \mathcal{T}(l, y, i, \mathtt{L})$ which by definition of the algorithm has happened — candidate r_1 has been placed. Therefore we can conclude that $\mathcal{T}(l, r_1, i+1, \mathtt{L})$ indeed contains a maximum pp-axis for the level $i+1$.

Part 3. The right-skewed case.

Since the *right-skewed* configuration is symmetric to the *left-skewed*, an analogous argumentation also applies for *right-skewed case*.

We now prove the claim that the algorithm returns a maximum set of candidates that induces a single-peaked election. By the construction of Algorithm 1, the last step iterates over all entries of $\mathcal{T}$ and returns a maximum pp-axis P. By Lemma 3.5, P contains the maximum number of candidates that induce a single-peaked election, proving the claim.

Finally we analyze the computational complexity of the algorithm through. The algorithm contains three major parts: the initialization (lines 2-4), the main loop (lines 5-13) and the return instruction (line 14). We analyze each part in turn.

During the initialization the algorithm sets every combination of l, r, z for level 0 to an initial value which can be done in $\mathcal{O}(m^2)$. Then it calls the procedure listed in the Algorithm 2 which executes in $\mathcal{O}(mn)$ steps. Altogether the initialization takes $\mathcal{O}(m^2 + mn)$ time.

We now analyse the main loop (lines 5-13). The outer loop in the line 5 is executed $\mathcal{O}(i_{max})$ times. Copying the values of the state table from the previous level in lines 6-7 can be done in $\mathcal{O}(m^2)$ steps. The loops in lines 8-10 and 11-13 are each executed $\Theta(|W_i|m^2)$ times. All operations from the main loop but the call to the PLACE procedure from Algorithm 3 can be performed in $\mathcal{O}(1)$ steps.

The time complexity of Algorithm 3 may not be immediately obvious. Due to Escoffier et al. [Esc+08] we known that a single-peaked axis check after inserting one new candidate x between the boundary can be done in $\mathcal{O}(n)$ steps because we know $(\ldots l \leftrightarrow r \ldots)$ is already single-peaked. For the fine details we refer to the original work.

We can now calculate the time complexity of the main loop:

$$\begin{aligned}
\mathcal{O}(\sum_{1<=i<=i_{max}} (m^2 + |W_i|m^2n)) &= \\
\mathcal{O}(\sum_{1<=i<=i_{max}} (m^2) + \sum_{1<=i<=i_{max}} (|W_i|m^2n)) &= \\
\mathcal{O}(\sum_{1<=i<=i_{max}} (m^2)) + \mathcal{O}(\sum_{1<=i<=i_{max}} (|W_i|m^2n)) &= \\
\mathcal{O}(m^3) + \mathcal{O}(\sum_{1<=i<=i_{max}} (|W_i|m^2n)) &= \\
\mathcal{O}(m^3) + \mathcal{O}(m^2n \sum_{1<=i<=i_{max}} |W_i|) &= \\
\mathcal{O}(m^3) + \mathcal{O}(m^2n|\mathcal{W}|) &= \\
\mathcal{O}(m^3) + \mathcal{O}(m^3n) &= \\
\mathcal{O}(m^3n) &
\end{aligned}$$

To return a value (line 14), the algorithm has to iterate over all entries of $\mathcal{T}$. Because of the Lemma 3.5 this can be done in $\mathcal{O}(m^3)$ steps which can be trivially reduced to $\mathcal{O}(m^2)$ by considering every combination of x, y, z only for i_{max}.

The total time complexity of the algorithm is then $\mathcal{O}(m^2 + mn + m^3n + m^2) = \mathcal{O}(m^3n)$. □

3.3 Notes on the algorithm

In this section we consider a couple of interesting properties of the algorithm and also possible improvements and extensions.

We found it really easy to overestimate the complexity of Algorithm 1 by applying loose upper bounds and conclude that the algorithm has $\mathcal{O}(m^4n)$ time complexity. Since we found this error interesting, we analyse it.

Let $\mathcal{W}$ be a worst candidate list with $\frac{m}{2} + 1$ worst candidate sets: $\frac{m}{2}$ of which contain only one candidate, while the remaining worst candidate set contains $\frac{m}{2}$ candidates. Consider again the main loop (lines 5-13) of the algorithm. The number of the iterations of the outer loop from line 5 can be upper bounded by $\mathcal{O}(m)$ since $\mathcal{W}$ contains $\Theta(m)$ worst candidate sets. Now the number of iterations of the loops starting in lines 8 and 11 can also be upper-bounded by $\mathcal{O}(m)$ since $\mathcal{W}$ contains a worst candidate set with $\Theta(m)$ candidates.
The bodies of the inner loops have $\mathcal{O}(m^2n)$ time complexity just as before. Multiply-

ing $\mathcal{O}(m)$ steps for the outer loop times $\mathcal{O}(m)$ for the loops starting in lines 8 and 11 times the $\mathcal{O}(m^2n)$ for the bodies of these loops results in $\mathcal{O}(m^4n)$.

Following this train of thought it is easy to overlook the reason for the overestimation: iterating over worst candidate sets and then iterating over the candidates from each set can only result in $\mathcal{O}(m)$ steps, as each candidate is represented in $\mathcal{W}$ only once.

Notice that it is possible to construct $\mathcal{W}$ to have $k \in \mathbb{N}^+$ worst candidate sets W_i with $\frac{m}{k+1} \in \Theta(m)$ candidates each, and $\frac{m}{k+1} \in \Theta(m)$ worst candidate sets w_i with one candidate each:

$$\mathcal{W} = (w_1, w_2 \dots w_{m/(k+1)}, W_{m/(k+1)+1}, W_{m/(k+1)+2} \dots W_{m/(k+1)+k-1})$$

Another interesting aspect of the algorithm presents itself, when we attempt to adapt the the algorithm to only detect single-peaked elections. By incorporating the assumption of single-peakedness into the algorithm, we store only one pp-axis per level, instead of maintaining the whole table $\mathcal{T}$. Furthermore, when processing each level, we either place all candidates from this level on the axis from the previous level, reducing the complexity further. If we are unable to place all candidates, then the election is not single-peaked. With these adjustments the algorithm becomes a single-peaked detection algorithm with the running time $\mathcal{O}(mn)$, similar to the algorithm by Escoffier et al.[Esc+08].

The algorithm we have presented has space complexity $\mathcal{O}(m^4)$. This can easily be improved, if we do not care about $\mathcal{T}$ but only about the final result (which is true in most, if not all cases). For the algorithm to work correctly, only two levels of $\mathcal{T}$ need to be kept: the current one and the previous one. The algorithm does not consider any entries beyond these levels. Using this insight we can easily improve the space complexity from $\mathcal{O}(m^4)$ to $\mathcal{O}(m^3)$.

Alternatively, the space complexity can be improved by reducing the size of the $\mathcal{O}(m^2)$ entries in the table. The key point to this optimization involves only storing a reference to some other pp-axis (or an empty axis which we allow as a special case) together with the information where to put at most two candidates. This obviously requires only $\mathcal{O}(1)$ memory cells for each table entry to store. Recall that all pp-axes in $\mathcal{T}$ are built iteratively, that is, an axis from level $i+1$ always builds upon an axis from level i and adds at most two candidates.

During the implementation we evaluated the two approaches and found out that using only the first optimisation for the space complexity was actually the best one from software engineering perspective: the implementation was much simpler and more intuitive.

We claim that is possible to further reduce the practical memory footprint by combining the two methods, but would not recommend it unless absolutely necessary (for example due to hardware or software limitations). The reference management of table entries becomes very complex and prone to subtle errors when keeping only the last two levels.

3.4 Conclusion

In this chapter we have presented and proved our $\mathcal{O}(m^3n)$ time algorithm for Single-Crossing Candidate Deletion. It improves upon an existing algorithm with $\mathcal{O}(m^3n)$ time by Erdélyi, Lackner, and Pfandler [Erd+13].

We have provided two alternative formulations — top-down and bottom-up for the algorithm and have discussed some important concepts of the algorithm, most notably the absence of a Delete operation.

We also included a discussion on interesting aspects of the algorithm, its analysis as well as practical considerations and optimisations.

This algorithm completes the set of six algorithms which are used implemented in Chapter 4 and used for the analysis of PrefLib in Chapter 5.

Algorithm Engineering 4

In this chapter we describe our practical experiences with the six problems from Chapter 2. We discuss implementing, testing, optimising and extending the algorithms from Chapters 2 and 3.

We also briefly discuss Algorithm Engineering as our method of choice and the extent to which we have used it.

We deal with the six problems in the following order: we first consider the detection problems, then consider Single-Peaked Candidate Deletion and Single-Crossing Maverick Deletion. We continue with a discussion on the NP-hard problems Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion.

This order of problems coincidentally matches the gradient of difficulties we have encountered during implementation of the respective algorithms and their computational complexity.

We finish the chapter with some practical insights and comments on our approach and Algorithm Engineering in general.

4.1 The Method of Algorithm Engineering

We haven chosen Algorithm Engineering method proposed by Peter Sanders [San09] as the basis of our approach for practical evaluation because of its properties and our previous experiences during the studies of computer science.

The defining aspect of the Algorithm Engineering framework lies in its cyclic structure (see Fig. 4.1). A typical, straightforward approach to algorithms involves a one-way process of theoretical design and analysis, followed by practical implementation and evaluation. The philosophy of Algorithm Engineering recognises a fatal weakness of this process: the information only flows in one direction. This creates an imbalance between the two aspects of an algorithm, devaluating its practical aspects. Sanders proposes instead that the insight gathered from practical experiences, based on real-world data, should be used as input for further theoretical improvement, thus ensuring a bidirectional flow of information.

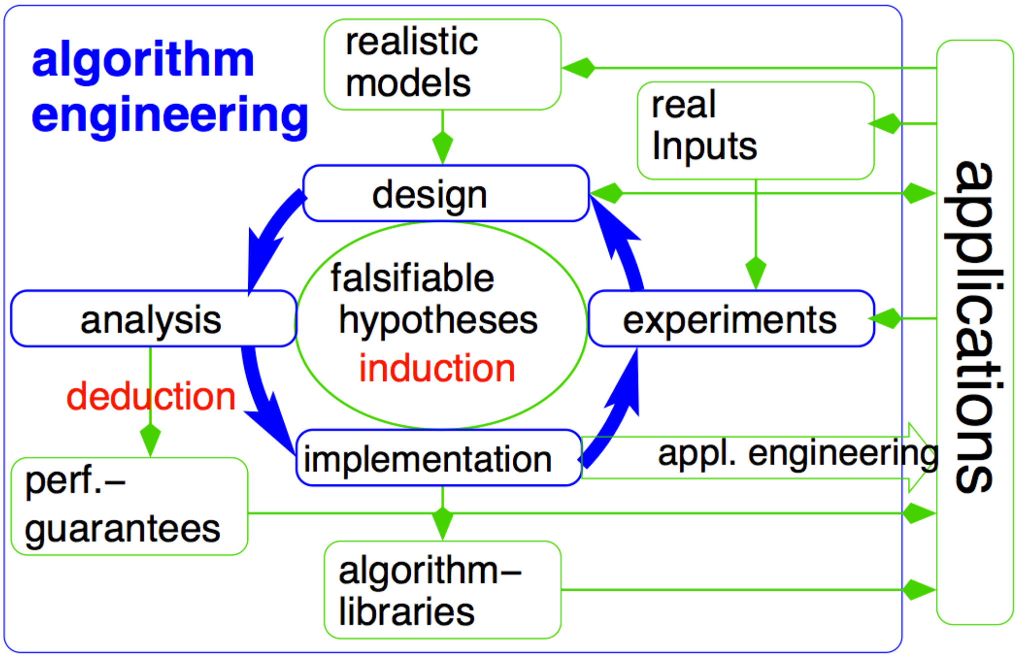

Fig. 4.1: Algorithm enginnering as proposed by Peter Sanders.

4.2 Experimental Setup

A crucial point to successful application of Algorithm Engineering is a proper consideration of real-world limitations like for example resource limits or hardware aspects of the target system and a choice of valid input data.

Our target system consisted of a single six-core Intel Xeon Processor X5650 running at a fixed frequency of 3 GHz, with disabled simultaneous multiprocessing (Hyper-Threading technology as it is called by Intel) to prevent core and cache thrashing (and thus interference). This processor has been paired with 16 gigabytes of system memory. As for the software platform, we have chosen Java 8 running on Microsoft Windows 10. Our decision to use Java as programming language and execution environment was motivated by a combination of two factors: our own proficiency and its popularity. The operating system has been deemed an insignificant factor and was chosen simply based on availability (most notably, we have had an immediate access to a licensed version of Gurobi ILP solver on this Windows system).

Each experiment would consist of an application of at least one algorithm on all instances that we have obtained from PrefLib [MW13]. An extensive discussion of the data can be found in Chapter 5. By including all available data in our experiments we sidestep the concerns of choosing a valid and representative set of data.

During each experiment we would execute up to six algorithms in parallel. Each algorithm has been assigned a time limit of 20 minutes per instance. This was necessary because two of the problems do not have practically efficient algorithms available and could dominate processing capabilities. Indeed our first attempt without time limits had to be aborted after fruitless 72 hours. The choice of 20 minute ceiling per instance-problem pair was not random: we have estimated that this time limit should result in an experiment finishing in 24 hours, a time window that we have judged to be striking the right balance between useful (in providing more information) and hindering (by delaying further work).

4.3 Detection Problems

The first question regarding PrefLib data that we have sought to answer was the number of instances which are single-peaked or single-crossing. Therefore our first practical goal was a practically correct implementation of single-peaked and single-crossing detection algorithms. We started with the assumption that the algorithms are already very efficient and practical. This required a slight tweak to our Algorithm Engineering approach: instead of focusing on improving the algorithms, we had to focus on practical correctness and completeness.

Single-Peaked Detection

The linear-time algorithm by Escoffier et al. [Esc+08] was an obvious candidate for single-peaked detection (see our remarks in Section 2.4). Not only had their paper a significant impact on computational social choice, but also their approach is significantly simpler than the alternative: implementation of PQ-Trees for CONSECUTIVE ONES problem [BT86].

The difficulties we have encountered during the implementation stem from the fact that Escoffier et al.. [Esc+08] did not give a compact or pseudo-code formulation of their algorithm and instead wrote it down descriptively. This allows for interpretation regarding two points of the algorithm.

The first problem is how to correctly deal with instances with only one initial worst candidate, followed by two candidates ranked second-to-last (see for example Fig. 4.2).

The authors glance over this detail leaving room for speculation. To correctly deal with this case, while implementing the algorithm as closely to its description as possible, we introduce a dummy candidate, which is ranked last by all voters. Obviously this does not change the result of the algorithm, but provides two reference

$$a \succ_1 b \succ_1 c \succ_1 d \succ_1 e$$
$$a \succ_1 b \succ_1 d \succ_1 c \succ_1 e$$
$$a \succ_1 c \succ_1 b \succ_1 d \succ_1 e$$

Fig. 4.2: Initial state with only one worst candidate is problematic for the algorithm by Escoffier et al. when there are two second-to-last candidates.

candidates for the correct execution of the PLACE procedure on the pair $\{c, d\}$. The dummy candidate is removed from the result after the algorithm terminates.

The second issue is a very minor one: the compact write-up of the algorithm would suggest that the PLACE procedure checks four cases when placing two candidates at a time, where in reality there are six cases (two additional ones through symmetry).

After amending the algorithm to close these gaps we were able to successfully execute the algorithm on all input instances. To eliminate false positives we have implemented a straightforward linear-time verification algorithm which executed in tandem with single-peaked detection algorithm.

A careful reader may notice that we did not address the problem of potential false negatives. Indeed the algorithm and verifier by themselves are insufficient in this regard. Not until we have had implemented further algorithms, most importantly for SINGLE-PEAKED CANDIDATE DELETION, we were able to leverage the n-version programming (NVP) [CA78] method to address false negatives.

The n-version programming method involves implementing different algorithms for the same problem and comparing them to eliminate both false positives and false negatives. It a legitimate approach to engineering of practically correct software [Avi95]. Ideally the two algorithms would be implemented by two different, geographically and culturally separated teams on a sufficiently diversified hardware. Given the constraints of the master thesis such measures were not only to extreme but also impossible to adhere to. In our case we have found running of the algorithm for SINGLE-PEAKED CANDIDATE DELETION and comparing its result with 0 completely sufficient and satisfactory.

Had we not implemented an algorithm for SINGLE-PEAKED CANDIDATE DELETION we could have also used a reduction to the CONSECUTIVE ONES problem instead for verification.

Single-Crossing Detection

We have chosen one of the $\mathcal{O}(m^2n)$ algorithms for single-crossing detection proposed by Bredereck, Chen and Woeginger [Bre+13a] for implementation. Of the two we have disregarded the one involving reduction to the CONSECUTIVE ONES problem, as it involves implementing the PQ-Trees algorithm. This has left us with the algorithm we have summarised in Section 2.4.

The algorithm has been extremely straightforward in implementation, the only challenge involved correctly handling the edge cases (outermost segments) when implementing the list-of-segments data structure.

Similarly to the single-peaked case we have employed a simple verification algorithm. Furthermore having implemented further polynomial-time algorithms we were able to use n-version programming to further validate results of single-crossing detection.

4.4 Polynomial-time Deletion Problems

After establishing a basic set of tools and considering the first results, we have decided to implement the known polynomial-time algorithms for measuring the distance to single-peaked or single-crossing profiles.

For the SINGLE-PEAKED CANDIDATE DELETION problem we have first considered implementing the established $\mathcal{O}(m^5n)$ algorithm, but as discussed in Chapter 3 we have settled on our own $\mathcal{O}(m^3n)$ algorithm.

Single-Peaked Candidate Deletion

Our theoretical work on the $\mathcal{O}(m^3n)$-time algorithm for the SINGLE-PEAKED CANDIDATE DELETION problem enabled us to swiftly and easily implement the algorithm. The only interesting aspects have already been discussed in Section 3.3: we preemptively (i.e. before running the experiments) reduced the complexity of the algorithm and its memory consumption by only keeping the required part of the state table in the system memory. We should also note that while the initial versions kept track of pp-axes for different configurations, the final version did not, as they are irrelevant for our evaluation.

Implementing this algorithm enabled us to do a cross check with the single-peaked detection algorithm. During development we have employed the single-peaked detection algorithm to ensure correctness at different points of the construction.

We were able to execute this algorithm on all input instances, hence no further changes to the design of the algorithm have been deemed neccessary.

Single-Crossing Maverick Deletion

For the SINGLE-CROSSING MAVERICK DELETION problem we employed the algorithm by Bredereck, Chen and Woeginger [Bre+13b]. Since their algorithm involves finding maximum-weight paths in a directed acyclic graph, we have decided to use an already established library for this purpose. We have settled on JGraphT (`http://jgrapht.org/`) because of its stability and good reputation.

After obtaining a longer description of the algorithm from the authors than the conference version provides[1], we were able to successfully implement the algorithm. Fully understanding the implications and the crucial role of the disagreement set is the only tricky part of the algorithm.

To our surprise we were unable to execute the algorithm on all instances successfully. This was caused by the experiment suite crashing due to memory exhaustion. This surprised us, because even with $\mathcal{O}(n^3)$ nodes and $\mathcal{O}(n^4)$ edges in the graph, we were not expecting to have any problems with memory consumption given the relative small size of the instances in PrefLib (about 2/3 of all instances have less than 10 unique preference profiles). We attribute this to the way JGraphT represents and stores nodes and edges that we have used in our implementation.

This proved to be an excellent opportunity to apply a second round of the Algorithm Engineering cycle: using the practical observations we had to change the algorithm to match the practical constraints.

Our approach has proved simple yet effective: instead of generating one graph with $\mathcal{O}(n^3)$ nodes, we generated $\mathcal{O}(n)$ graphs with $\mathcal{O}(n^2)$ nodes instead. Recall from the discussion in Section 2.4 that (almost) every node in the graph represents a pair of voters (v_1, v_2) and an edge between two such nodes is only considered if they share the first voter of the pair. Thus the original graph is more akin to a forest of trees, each rooted in a (v, v) node (for some v). Since a maximum path can only run through one of the trees, we can consider them separately.

This change does not impact the time complexity of the algorithm, but was sufficient to address our memory problems due to reduced space complexity.

[1] At the time of the writing of this thesis, a longer version of the paper by Bredereck, Chen and Woeginger [Bre+13b] was under review for journal publication.

Similarly to the previously described algorithms, we applied cross checking between this and the detection algorithm.

4.5 NP-Hard Deletion Problems

After completing the four polynomial-time algorithms we have decided to complete the set of the six problems and attempt to build a practical solution for the NP-hard problems Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion.

In this case the Algorithm Engineering framework seemed like an ideal choice: using the number of solvable instance we would be able to continue the cycle until a satisfactory result is reached.

The two problems may seem very different from a theoretical point of view, but from a practical one they are very similar in regards to strategies used to deal with them. Therefore we do not consider them separately.

In the following sections we document three iterations of the Algorithm Engineering cycle during which we attempted different approaches to tackling the NP-hard problems.

Search-Tree Algorithms

Our first approach when dealing with FPT problems is to formulate a search-tree algorithm for them.

For the single-peaked case we used the following simple recursive algorithm: given a parameter k and an election, find a forbidden substructure with two (α-configuration) or three voters v_1, v_2, v_3 (worst-diverse configuration). Remove v_1, decrement k by the weight of v_1 and call the procedure recursively. If a result is found for the subproblem, then return it, appending v_1. Otherwise try with v_2 and finally with v_3. In cases when k would be less than 0 the procedure for the subproblem is aborted. Lastly, when no forbidden substructures exist, the procedure returns an empty set.

To guarantee a globally optimal solution the algorithm is executed iteratively with increasing values of k, starting with 0. It should be obvious that this algorithm is correct: it extensively searches through the problem space and eliminates forbidden substructures until none are left, aborting should k become negative.

For the single-crossing case the algorithm is almost identical in its structure, but finds δ- and γ-configurations and branches into up to six recursive calls by removing candidates from the forbidden substructure instead.

This simple and naive approach proved fruitless and was the reason for introducing time limits on execution. After 72 hours we were unable to gather meaningful results.

After introducing time constraints we were still dissatisfied with the number of solved instances. This prompted us to rethink and redesign our approach, because the experiments have shown that naive search tree algorithms cannot be practically used in this case.

d-Hitting-Set Reduction

After the fiasco of the naive search-tree algorithms, we identified two possible ways forward: the first involved improving the search-tree algorithms and applying kernelization and the second involved the reduction to a d-Hitting-Set problem.

We have decided to use a d-Hitting-Set reduction for both problems as proposed by Elkind and Lackner [EL14]. The Hitting-Set instance would be then fed to a commercial ILP Solver. In our case we had access to the Gurobi solver, which is one of the leading tools. This contrasts with the results of Elkind and Lackner who have employed the state-of-the-art FPT algorithms for d-Hitting-Set family of problems [Che+10; Wah07; Fer10]. The reasoning behind our choice is simple: using an ILP Solver as a black-box for a hitting-set problem is straightforward, while the FPT algorithms are extremely clever and sophisticated and thus difficult to implement correctly in practice. Furthermore there are not guaranteed to be faster in practice than an ILP solver.

Recall that the reduction by Elkind and Lackner involves enumerating all forbidden substructures as sets (of candidates or voters) for the Hitting-Set algorithm.

This approach also proved completely infeasible: for nontrivial instances we were unable to completely enumerate all forbidden substructures within the time limit (!), which prevented us from gathering data without even attempting to solve the respective instances.

Iterative d-Hitting-Set Reduction

This flaw seemed fixable and a solution within reach: instead of iterating over all forbidden substructures, we would iteratively call the solver. This approach could be seen as a combination of the search-tree strategy and the Hitting-Set reduction. The core approach still involved a Hitting-Set reduction, but in a more clever way.

Our algorithm first finds a forbidden substructure, generates a set of constraints for it, and feeds a Hitting-Set instance with these constraints to a solver. The results are interpreted as candidates or voters which are then deleted from the election. The algorithm continues and tries to find another forbidden substructure - if it succeeds, then it generates a set of constraints for it and adds them to the ones previously generated. The solver is executed again, this time with a larger set of constraints. This continues until no further forbidden substructures can be found.

Using this approach we were finally been able to achieve satisfactory results. We were able to solve 99% instances for the Single-Peaked Maverick Deletion[2] and 75% instances for the Single-Crossing Candidate Deletion problem.

The difference in percentages can be explained using two facts. Most instances have very few unique preference profiles (about 2/3 have less than 10), but there are many instances with significant number of candidates (22% of the instances have more than 30 candidates). Secondly, candidate deletion seems "harder" because its associated forbidden substructures can have up to six candidates which have to be considered (resulting in a 6-Hitting-Set), which is double of the number of voters for the single-peaked case (which gives a 3-Hitting-Set).

4.6 Conclusion

In this chapter we have discussed our experiences with implementing the algorithms for the six problems in restricted domains. We have chosen the Algorithm Engineering framework as our approach.

For the three problems: Single-Peaked Detection, Single-Crossing Detection, Single-Peaked Candidate Deletion we had no real need for Algorithm Engineering. The algorithms performed correctly and within given constraints after just one iteration of the "cycle".

The benefit of Algorithm Engineering started to show when faced with resource limitations during the testing of the algorithm for Single-Crossing Maverick Deletion. We note that any reasonable person would employ a similar strategy to ours, with or without the explicit intent of Algorithm Engineering.

[2] We able to solve only 96% of instances for Single-Peaked Maverick Deletion after setting the weights to one. This effect persisted throughout multiple executions of the experiment.

The real value became obvious when confronted with problems having no widely researched and practical solutions: the NP-hard SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING CANDIDATE DELETION. In these cases the discipline of working under Algorithm Engineering framework allowed us to move forward by early change of direction and refinement of our working hypothesis when faced with technical difficulties.

As for purely algorithmic observations we would recommend to use our improved version of the algorithm for SINGLE-CROSSING MAVERICK DELETION, since it has the same time complexity, yet better space complexity as the original, and it is expected to perform much better in practice. Furthermore we believe that our results regarding NP-hard problems can be improved by applying data reduction rules and further clever tricks (like for example rounding of approximate solutions).

5 Results

In this chapter we describe the application of the algorithms developed in the Chapter 4 on the data provided by PrefLib. We begin with the discussion of the methodology and how it fits with other aspects of our thesis. We then discuss the data sets provided by PrefLib. We proceed to evaluate each data set separately.

We conclude the chapter with general observations regarding all the available instances, recommendations for future work and consequences for other practitioners working with restricted domains.

5.1 Methodology

For the evaluation we gathered data using almost identical method to the one employed in Section 4.2. We apply each of the six algorithms to each of the instances for at most 20 minutes. Instances that would take longer than 20 minutes are deemed unsolvable. This may skew results for Single-Peaked Maverick Deletion and Single-Crossing Candidate Deletion, more so for the latter problem, as it seems to be "harder" problem in practice than the others. We accept this source of error, as in our opinion it would be extremely surprising if the missing data points should change the results of our analysis drastically.

In this chapter, we will use the term "performance" meaning the remaining percentage of candidates (or voters) after application of the respective algorithms.

Since no instances were found to be single-peaked or single-crossing, we adapt our research questions accordingly. We seek to answer the following questions:

1. How reasonable it is to apply the label of "nearly single-peaked" to the data in PrefLib using either of the two distance measures?

2. How reasonable it is to apply the label of "nearly single-crossing" to the data in PrefLib using either of the two distance measures?

3. Is the data "more" single-peaked than single-crossing?

4. Does considering only unique profiles provide a lower bound on the distances to single-peaked or single-crossing profiles?

We pose the questions purposely using a vague language, because it is hard to quantify what is reasonably nearly single-peaked (or single-crossing) — surely this can be said of removing 10% voters or 10% candidates, but what about 20%, or 33%? We believe that the concrete, quantifiable answer will depend on the exact field of application. Our goal is to provide a general impression of the results and allow the reader and the practitioners to draw their own conclusions.

The second to last question is easier to quantify. An instance is obviously "more" single-peaked if to achieve a single-peaked profile requires removing less candidates (or voters) than to obtain a single-crossing profile. We draw this comparison only for cases where both data points are available, i.e. where the algorithms for SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING CANDIDATE DELETION returned a result within the allotted 20 minutes.

The last question is the most theoretical one: as some instances contain weighted profiles, we also consider the variants of SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING MAVERICK DELETION for only unique profiles, i.e. we ignore the weights from the instance and arbitrarily set all the weights to one. We are interested in whether this process gives a lower bound on the performance for maverick deletion on the data from PrefLib.

On some charts we will present two variants of maverick deletion problems; one set of markers represents normal results (marked with "weighted") and the represents the same problem but weights of preference profiles set to one (marked with "unique", i.e. preference profiles with pairwise distinct preference orders). This distinction is important especially in the discussion of the last question stated above.

We expect to find more nearly single-peakedness to be more common than nearly single-crossigness. We base this expectation on the combinatorial fact, that for random instances single-peaked profiles should be much more common than single-crossing profiles (there are $\mathcal{O}(2^m)$ [Esc+08] single-peaked profiles and $\mathcal{O}(m^2)$ [Bre+13a] single-crossing profiles). We thus also expect to single-peaked algorithms to return larger instances.

Furthermore we believe that computing maverick deletion problems for elections with weights set to one will provide a lower bound for the results on weighted profiles.

The charts contain only the results that we were able to compute, comparing SINGLE-PEAKED MAVERICK DELETION to SINGLE-CROSSING MAVERICK DELETION, and SINGLE-

Peaked Candidate Deletion to Single-Crossing Candidate Deletion. This may be confusing and misleading, even more so given our ordering of the instances. To allow direct comparison of the performance on each instance, we provide comparison charts where applicable. In these charts results for every instance are compared with each other (an "apples-to-apples" comparison) and only for instances where both data points are available.

5.2 Data sets

The data provided by PrefLib consists of 314 instances, which belong to one of eight categories. As stated before we only consider instances containing strict total orders as preferences. All descriptions in this section which are in quotes come directly from the PrefLib website.

The first data set with the designation ED-00004 contains Netflix Prize Data. It is based on competition called the Netflix Prize which aimed to improve the accuracy of the movie recommendations on the streaming platform provided by Netflix. As stated by PrefLib the data consists of two groups: one with 1000 randomly chosen instances with 3 candidates and 6 unique profiles and a second one with 100 randomly chosen instances with 4 candidates and 24 unique profiles from the study by Mattei, Forshee and Goldsmith [Mat+12]. The number of voters for these instances ranges from few hundred to around 14 thousand.

A careful reader will notice that this data set provides a majority of available instances, and thus may skew the general observations as the movie data is overrepresented. We are aware of this problem and therefore have to decided to analyse all data sets separately. Our general analysis also makes note of this fact.

The second data set with the designation ED-00006 "contains figure skating rankings from various competitions during the 1998 season including the World Juniors, World Championships, and the Olympics. These data sets generally have 10-25 candidates (skaters) and 8-10 judges (voters)". We note the fact that all preference profiles are unique for all 19 instances from this data set. PrefLib website is missing the information on the submitter of this data set.

The third data set with the designation ED-00009 contains only two instances describing the results of "surveying students at AGU University of Science and Technology about their course preferences." The first instance contains 9 candidates with 146 voters (and 123 unique profiles). The second instance contains 7 candidates with 153 voters (and 70 unique profiles). This set has been submitted by Piotr Faliszewski from AGH University of Science and Technology.

The forth data set with the designation ED-00011 contains data comparing ranking of search results among five prevailing search engines. The instances contain either 103 or 240 candidates with 5 voters. This set has been submitted by Robert Bredereck from TU Berlin.

The fifth data set with the designation ED-00012 "contains complete rank orderings of T-Shirt designs voted on by members of the Optimization Research Group at NICTA. There are 11 designs (candidates) and 30 votes about these designs. Voters were required to submit complete strict orders.". It consists of a single instance submitted by Carleton Coffrin.

The sixth data set with the designation ED-00015 contains data similar to the ED-00011 set and consists of 78 instances with 4 or 5 voters and number of candidates ranging from 10 to 240. It has been submitted by Rober Bredereck as well. We note the fact that this is the second major contribution to the data available at PrefLib.

The seventh data set with the designation ED-00024 contains four instances with 4 candidates, 24 unique profiles and around 800 voters each. The data has been gathered by Andrew Mao using The Mechanical Turk service by Amazon. "Each of the candidates correspond (sic!) to random dots presented to a user on Mechanical Turk, who is asked to rank the items from those containing the least dots (first) to those containing the most dots (last). Thus, for all of these data sets there is a ground truth ranking which corresponds to the candidate names in sorted order. In the Dots task, each task contains elements with 200, $200+i$, $200+2i$, and $200+3i$ dots, where $i = 3, 5, 7, 9$. This allows for more noise to be introduced to various iterations of the task. For each i, 40 sets of puzzles were placed on Mechanical Turk and were ranked by 20 users. As per the data owners request these 160 individual trails have been aggregated into a single file for each i."

The eighth and last data set with the designation ED-00025 has an extremely similar construction to the set ED-00024. The difference lies in the underlying problem stated to the "mechanical turks". "Each of the candidates correspond (sic!) to an instance of the sliding puzzle game presented to a user on Mechanical Turk, who is asked to rank the items from those in a position closest to solution (first) to those requiring the most moves to complete (last). Thus, for all of these data sets there is a ground truth ranking which corresponds to the candidate names in sorted order. In the Puzzle task, each task contains elements requiring d, $d+3$, $d+6$, and $d+9$ moves to complete, where $d = 5, 7, 9, 11$. This allows for more noise to be introduced to various iterations of the task. For each i, 40 sets of puzzles were placed on Mechanical Turk and were ranked by 20 users. As per the data owners request these 160 individual trails have been aggregated into a single file for each i."

5.3 Experimental results

For better visual presentation we have ordered the instances by the number of candidates (for maverick deletion) and by the number of voters (for candidate deletion). Where applicable we used the percentage of remaining voters (respectively candidates) as a secondary criterium for ordering of the instances.

ED-00004 — Netflix Prize Data

For both SINGLE-PEAKED CANDIDATE DELETION and SINGLE-CROSSING CANDIDATE DELETION only elections with two remaining candidates provided a single-peaked or single-crossing profile. This is noteworthy because a election with only two candidates is by definition both single-peaked and single-crossing. For the instances with four candidates we doubt the usefulness of this result, as it involves deleting half of the candidates. In case of three candidates the result may seem more meaningful, as it involves deleting "only" one third of candidates. Nevertheless we doubt the practicality of this result.

The SINGLE-PEAKED MAVERICK DELETION and SINGLE-CROSSING MAVERICK DELETION provide a much different picture. Especially for the instances with only three candidates the results seem almost single-peaked (or single-crossing) as can be seen in Fig. 5.1 and Fig. 5.2. This lends to support to the "maverick interpretation" of maverick deletion problems, i.e. there is a (relatively) small set of nonconforming voters, a form of noise in the data.

We found it surprising that both algorithms produced instances of similar size, with median performance of single-peaked algorithm being 11% better than the single-crossing one, as can be seen in Fig. 5.3. It should be noted that this result carries a lot of variance.

For instances with four candidates the results are much worse and are around 50% for both problems. As expected, considering profiles with weights set to one gives a lower bound on general performance — for three candidates it is 66.6%, whereas for four candidates a much worse 33.3%. The results in this case were identical for both single-peaked and single-crossing maverick deletion.

ED-00006 — Figure Skating

This data set provided meaningful results for both SINGLE-PEAKED CANDIDATE DELETION and SINGLE-CROSSING CANDIDATE DELETION, as they were no resulting instances with less than 3 candidates, as depicted in Fig. 5.4 and Fig. 5.5. We

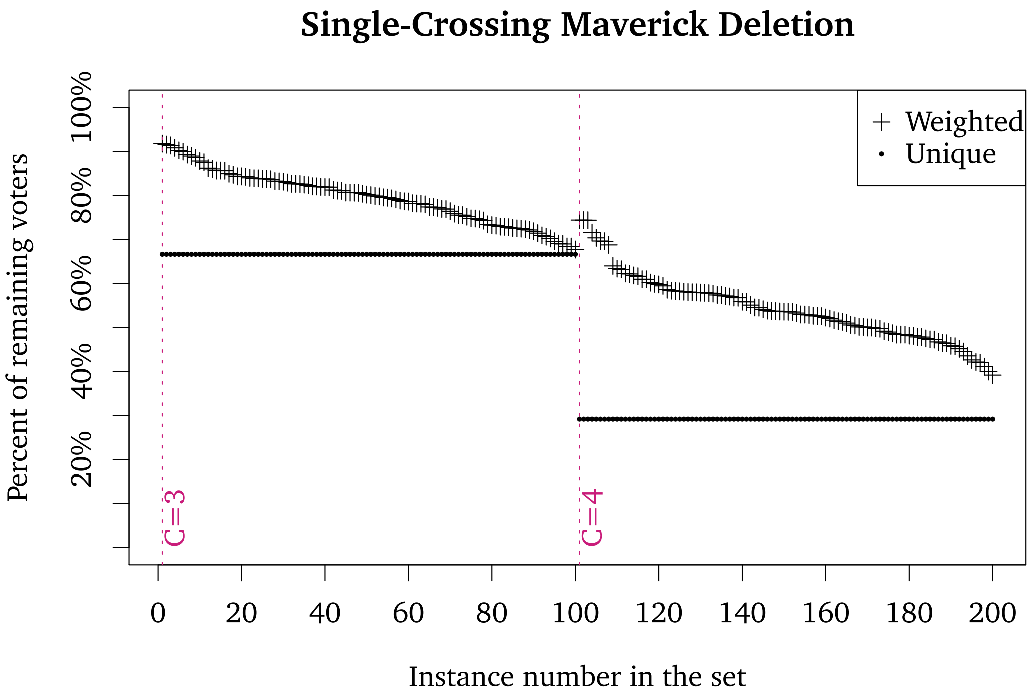

Fig. 5.1: Single-Crossing Maverick Deletion on Netflix Prize Data

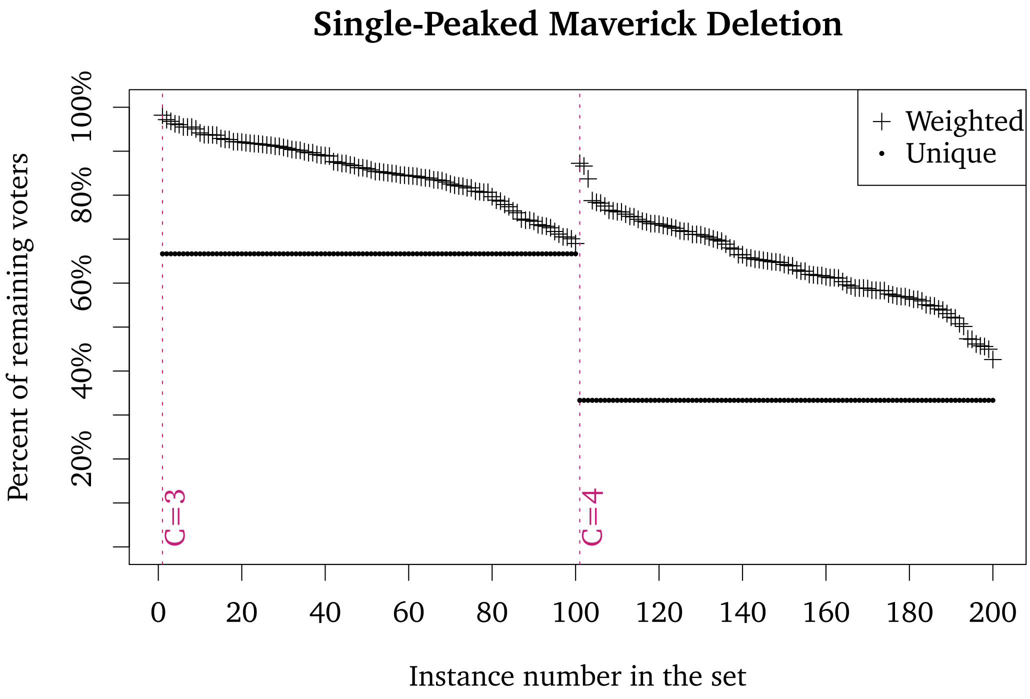

Fig. 5.2: Single-Peaked Maverick Deletion on Netflix Prize Data

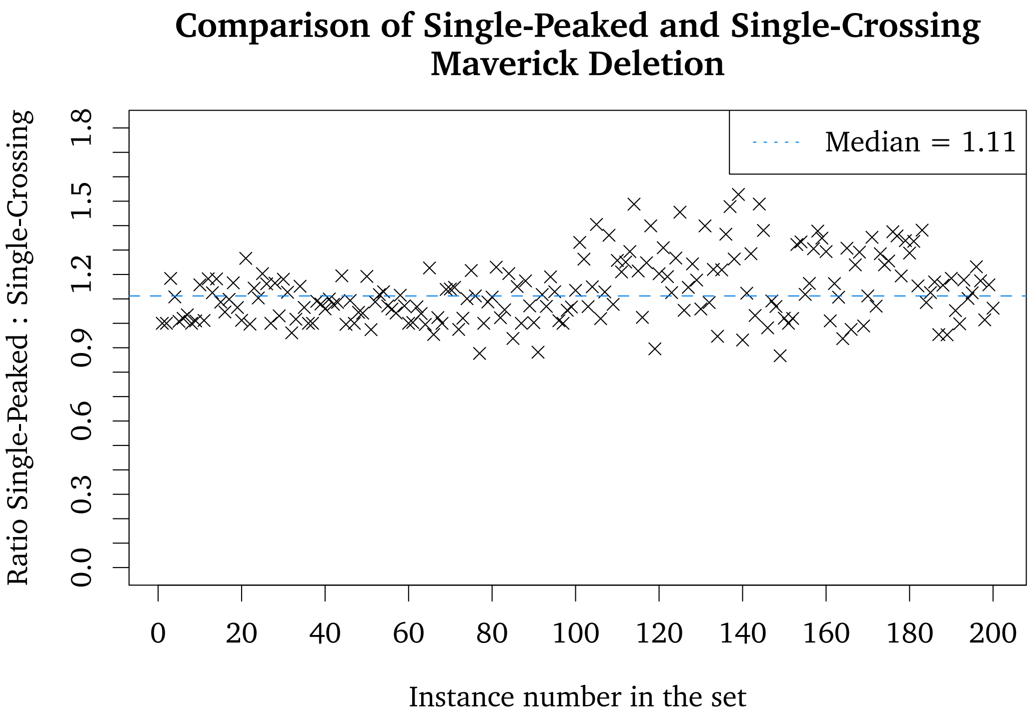

Fig. 5.3: Comparison of maverick deletion performance on Netflix Prize Data

note much better average and outlier performance for SINGLE-PEAKED CANDIDATE DELETION, which becomes more apparent in direct comparison (see Fig. 5.6) — the median result for SINGLE-PEAKED CANDIDATE DELETION is 18% better than it's counterpart. In our opinion it is reasonable to conclude that using SINGLE-PEAKED CANDIDATE DELETION the data set can be described as nearly single-peaked.

For maverick deletion problems the situation is reversed: except for two outlier instances no other instance achieve over 50% remaining voters in a single-peaked (Fig. 5.7) or a single-crossing profile (Fig. 5.8). The only interesting observation here is the consistency of the results: both algorithms performed almost identically, as depicted in Fig. 5.9. Since the instances contain mostly unweighted profiles, we abstain from drawing conclusions regarding lower bounds.

ED-00009 — Course Preferences

For the course preferences both algorithms for candidate deletion retained 33.3% of candidates from the first instance and 43% of candidates from the second instance, well below any reasonable threshold. This resulted in instances containing exactly three candidates.

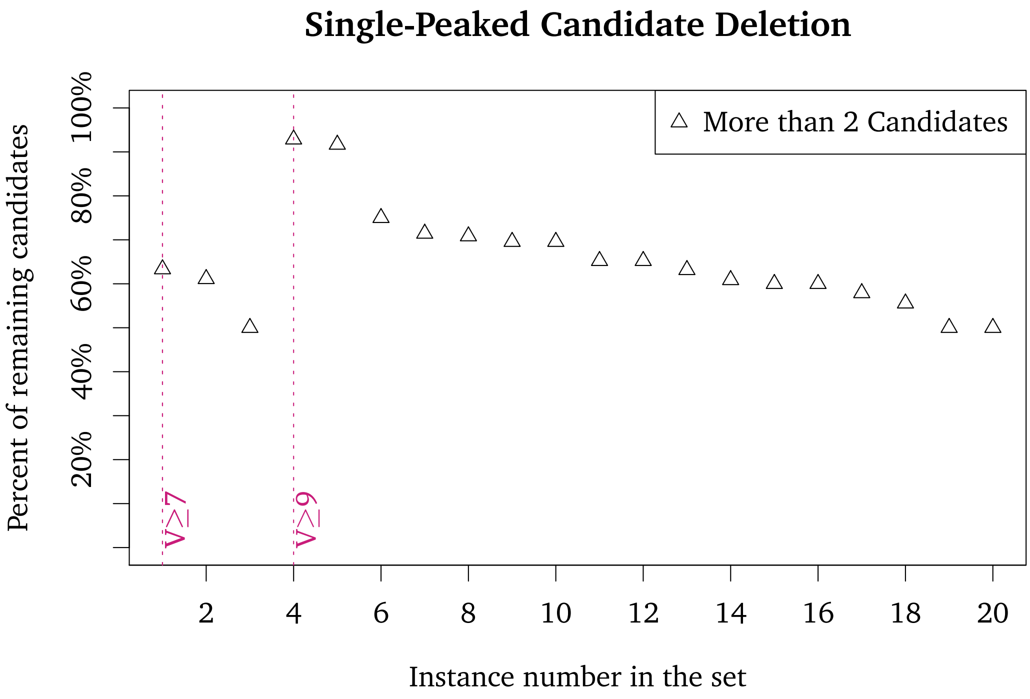

Fig. 5.4: Single-Peaked Candidate Deletion on Figure Skating Data

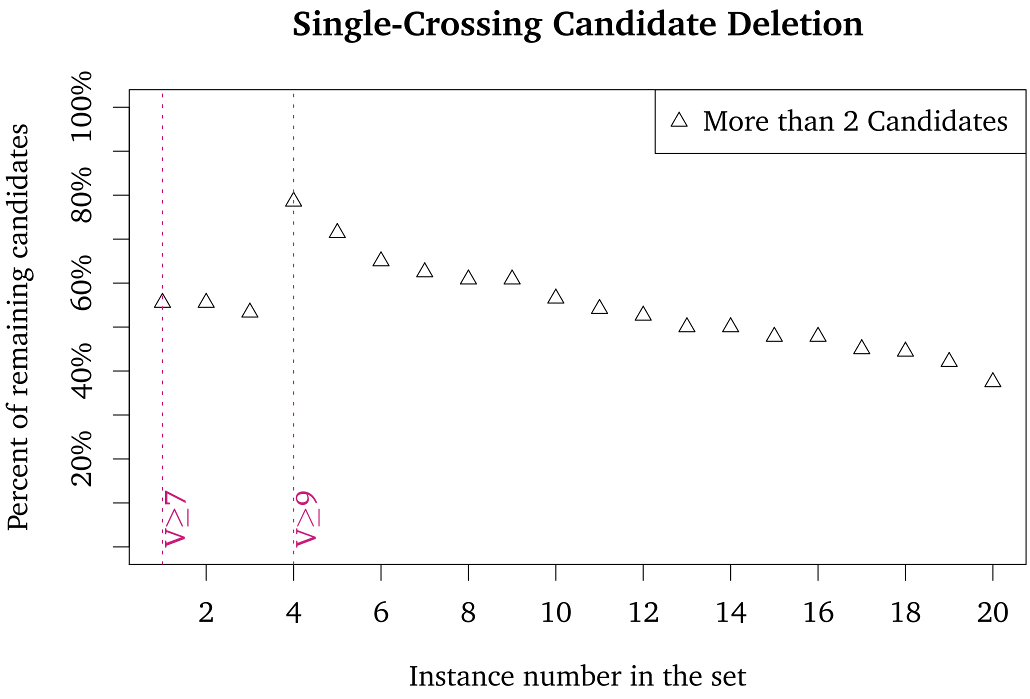

Fig. 5.5: Single-Crossing Candidate Deletion on Figure Skating Data

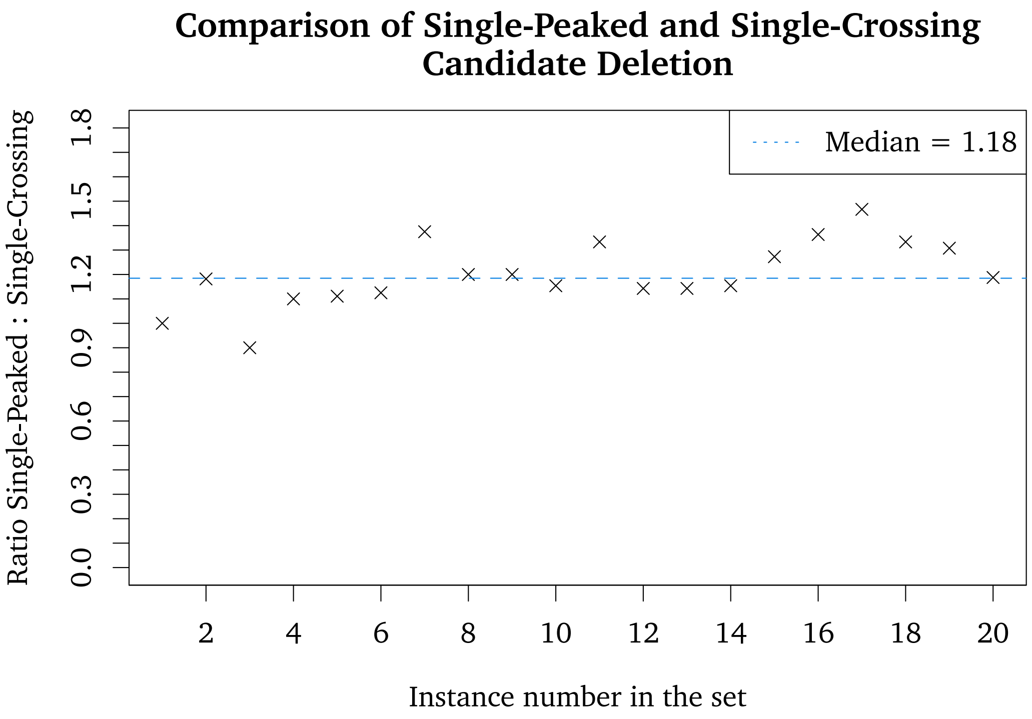

Fig. 5.6: Comparison of candidate deletion performance on Figure Skating Data

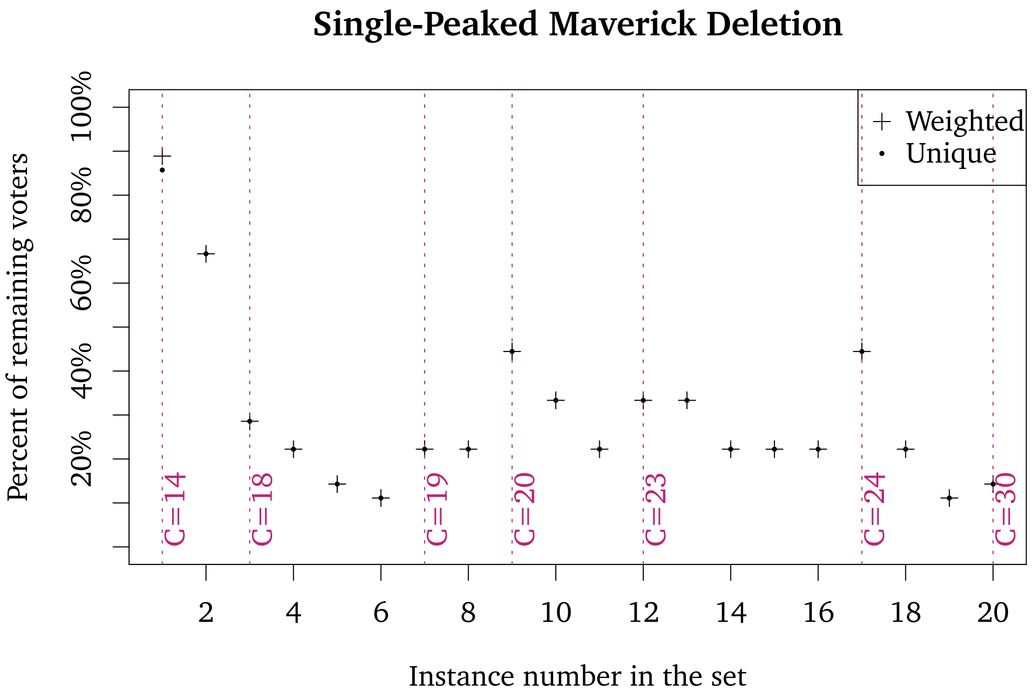

Fig. 5.7: Single-Peaked Maverick Deletion on Figure Skating Data

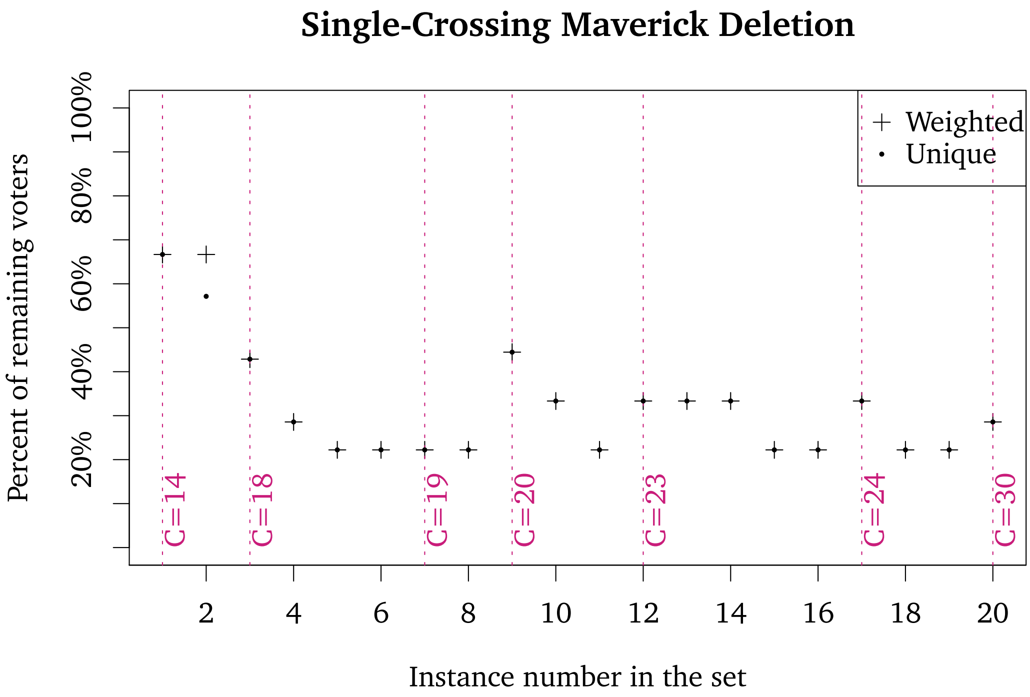

Fig. 5.8: Single-Crossing Maverick Deletion on Figure Skating Data

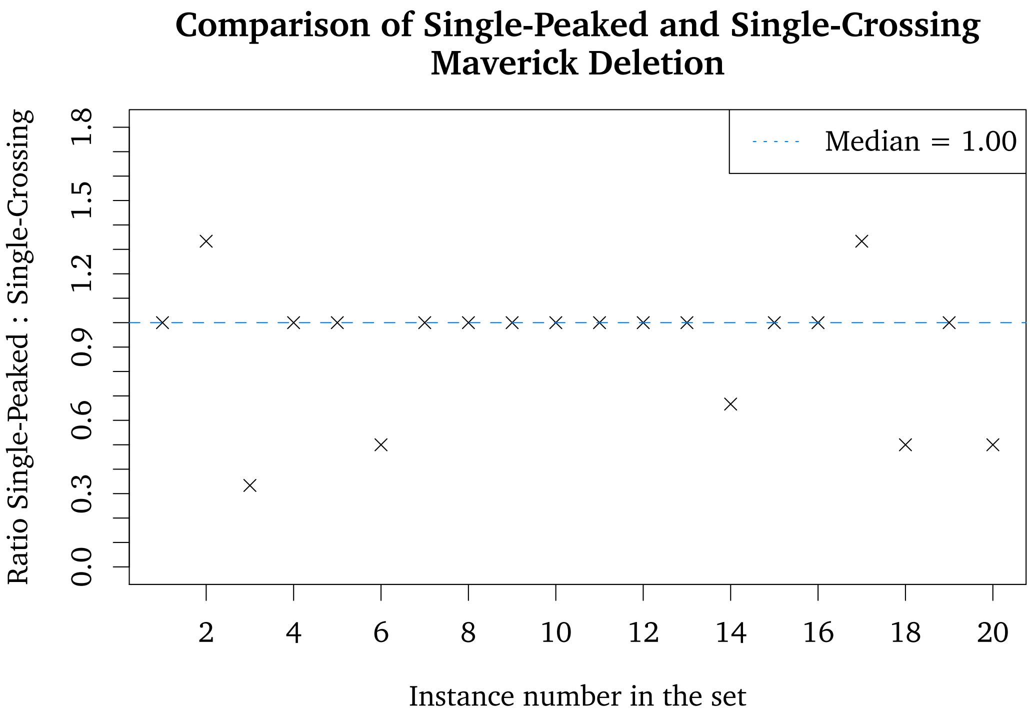

Fig. 5.9: Comparison of maverick deletion performance on Figure Skating Data

Our algorithm was unable to provide results for SINGLE-PEAKED MAVERICK DELETION. SINGLE-CROSSING MAVERICK DELETION deleted two thirds of voters from the first instance and almost 90% of the voters from the the second instance.

In our opinion one could reasonably argue that this data set is neither nearly single-peaked nor nearly single-crossing.

ED-00011 — Search Results 1

The results for the first search result set have been summarised in the table in Fig. 5.10. We note the absence of results for SINGLE-CROSSING CANDIDATE DELETION. We were shocked to observe that SINGLE-PEAKED MAVERICK DELETION required deletion of all but one voter.

Instance	Candidates	Profiles	SP-CD	SP-MD	SC-MD
1	103	5	14.6%	20%	40%
2	240	5	14%	20%	40%
3	242	5	7.4%	20%	40%

Fig. 5.10: Results for Search Results 1

Altogether this data set does seem to contain instances neither nearly single-peaked nor nearly single-crossing.

ED-00012 — T-Shirt Designs

This data set contains only one instance. We have summarised the results for it in Fig. 5.11. Note the absence of results for SINGLE-PEAKED MAVERICK DELETION for this instance, as we were unable to compute them given the constraints of our methodology.

Instance	Candidates	Profiles	SP-CD	SC-CD	SC-MD
1	11	30	27%	27%	10%

Fig. 5.11: Results for T-Shirt Designs

We think it is safe to conclude that this instance is neither nearly single-peaked nor nearly single-crossing.

ED-00015 — Search Results 2

Because of the abysmal performance of ED-00011 we were not expecting much better for ED-00015, as this set of instances intuitively feels similar. Sadly, for 95%

instances of this set we were unable to compute the results for SINGLE-CROSSING CANDIDATE DELETION.

To our surprise the maverick deletion algorithms performed better than for ED-00011 as depicted in Fig. 5.12 and Fig. 5.13. Still in most cases the resulting instances retained only one or two voters. It should be noted that single-peaked profiles were on average about 50% smaller than single-crossing profiles.

The results for SINGLE-PEAKED CANDIDATE DELETION are harder to interpret: we see that the data set can be partitioned in two classes: one with an average performance a little under 20% and the other with an average performance around 30% (Fig. 5.14)

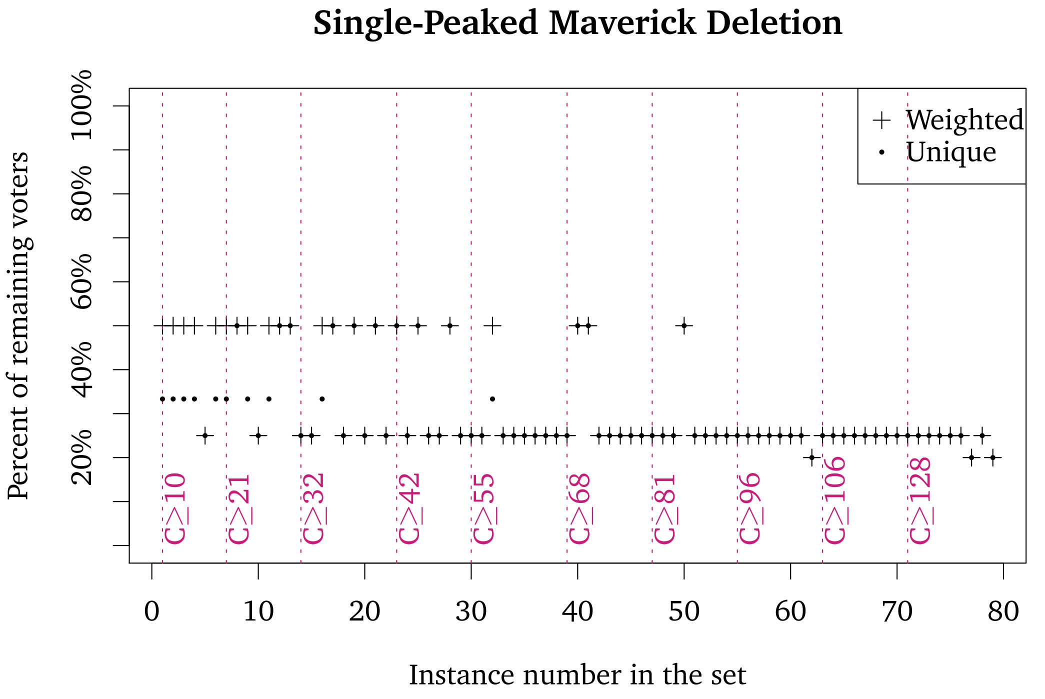

Fig. 5.12: Single-Peaked Maverick Deletion on Search Results 2

One could make a case for this data being nearly single-crossing using maverick deletion, but given the sizes of input instances (4-5 voters) and output instances (mostly 2-3 voters), we don not agree with this sentiment.

ED-00024 — Mechanical Turk 1

For this data set both SINGLE-PEAKED CANDIDATE DELETION and SINGLE-CROSSING CANDIDATE DELETION resulted in instances with only two candidates; i.e. removing

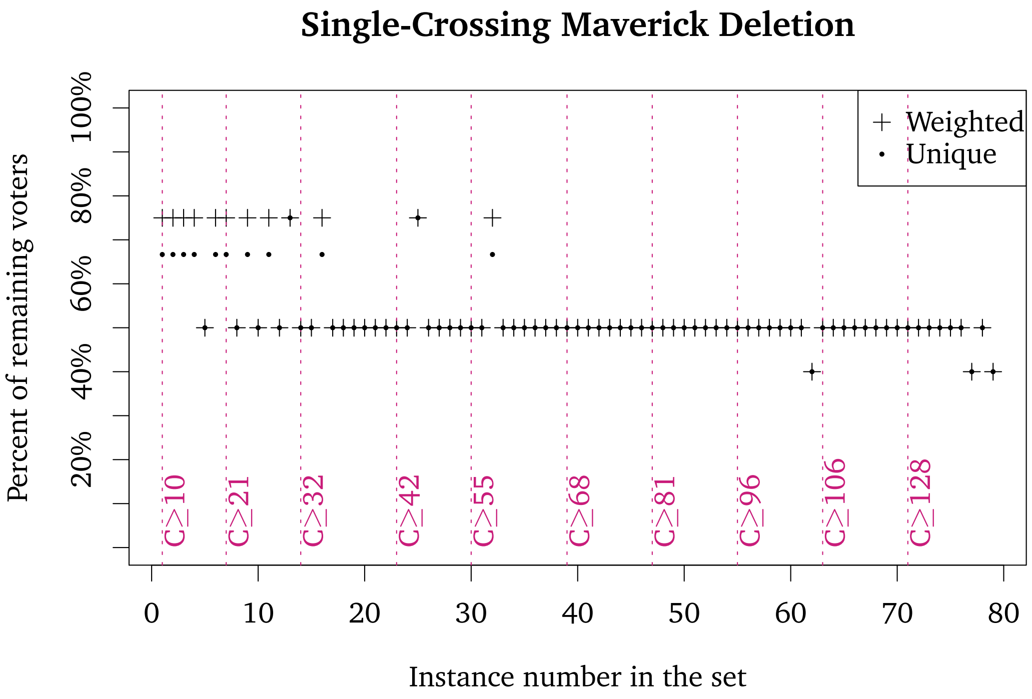

Fig. 5.13: Single-Crossing Maverick Deletion on Search Results 2

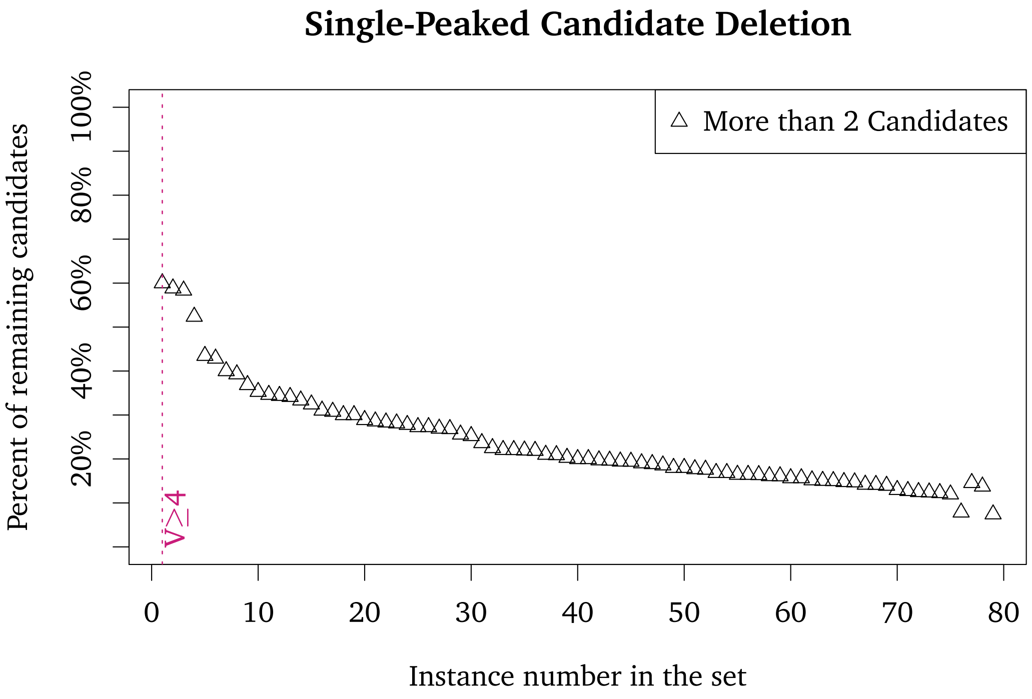

Fig. 5.14: Single-Peaked Candidate Deletion on Search Results 2

half of the original candidates. We think that our statement doubting meaningfulness of such result, stands here as well.

Application of SINGLE-PEAKED MAVERICK DELETION resulted in retaining of 44% to 62.5% of voters, which was on average about 40% better than SINGLE-CROSSING MAVERICK DELETION which retained 36% to 44% voters. We note the coincidence of upper and lower bounds of the problems.

In our opinion neither nearly single-peakedness nor nearly single-crossingness applies for the instances in this set.

ED-00025 — Mechanical Turk 2

The results from this set are very similar to ED-00024. As with the data set ED-00024 — Mechanical Turk 1, candidate deletion results in instances with only two candidates.

Maverick deletion problems performed similarly in the ranges 47% to 61% (single-peaked) and 38% to 48% (single-crossing). The difference in performance was smaller in this case; equal to around 26% on average. We again note the coincidence of upper and lower bounds of the problems.

In our opinion neither nearly single-peakedness nor nearly single-crossingness applies to the instances in this set.

General Observations

An aggregated view of the data provides couple of interesting insights: the results for candidate deletion problems are dominated by outputs with only two candidates — 66% instance for single-peaked profiles and 88.5% for single-crossing profiles (see Fig. 5.15 and Fig. 5.16).

It could be thus argued that in general, deleting candidates is not a good strategy when dealing with data from PrefLib, mostly because the available instances contain few candidates to begin with. In cases where the output contained more than 2 candidates the results are less clear: depending on the exact data set the instances could be judged nearly single-peaked or not.

The plots for maverick deletion problems are more interesting and clearly show three classes of instances: one with 3 candidates, one with 4 candidates and one with above 18 (or 15) candidates (Fig. 5.17 and Fig. 5.18). We believe this might be

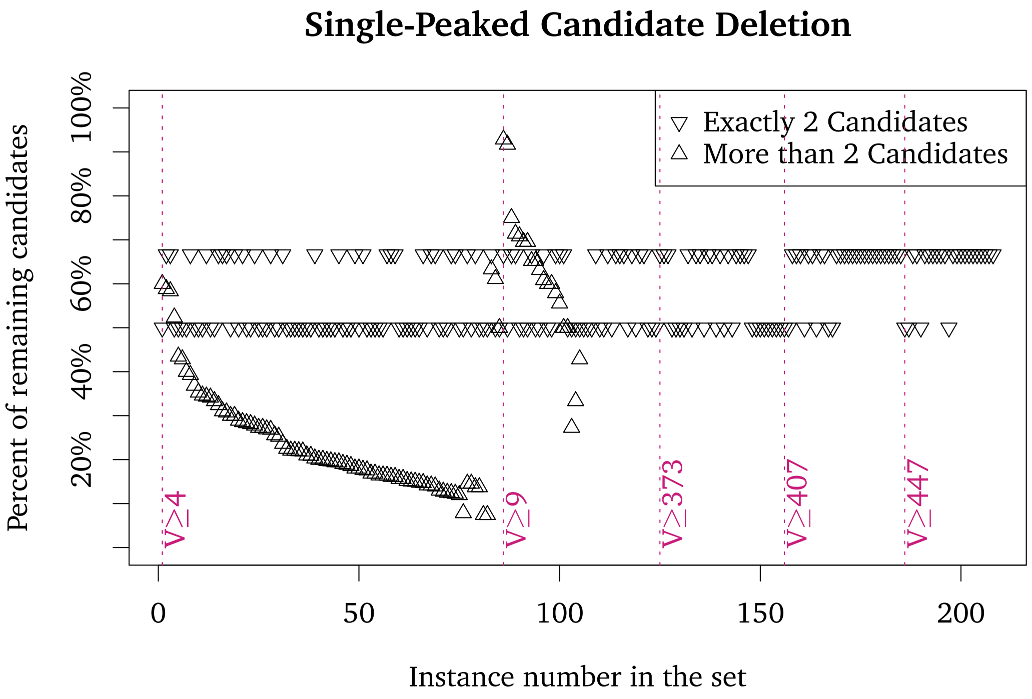

Fig. 5.15: Aggregated Results for Single-Peaked Candidate Deletion

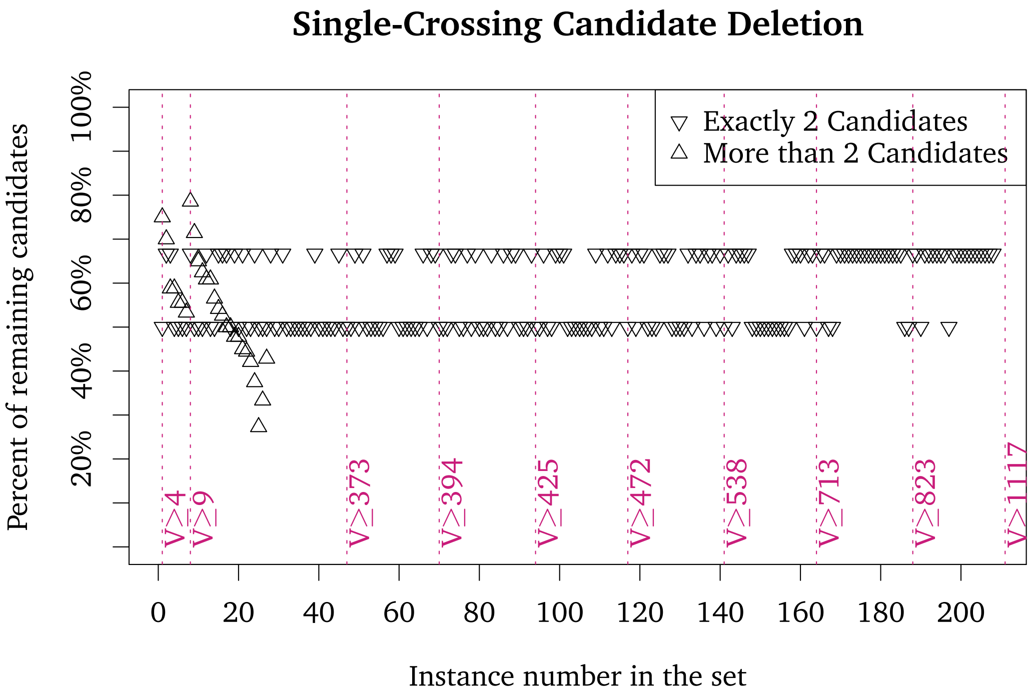

Fig. 5.16: Aggregated Results for Single-Crossing Candidate Deletion

an artifact of the overrepresentation of Netflix Prize Data. By excluding this data set this nice partitioning disappears.

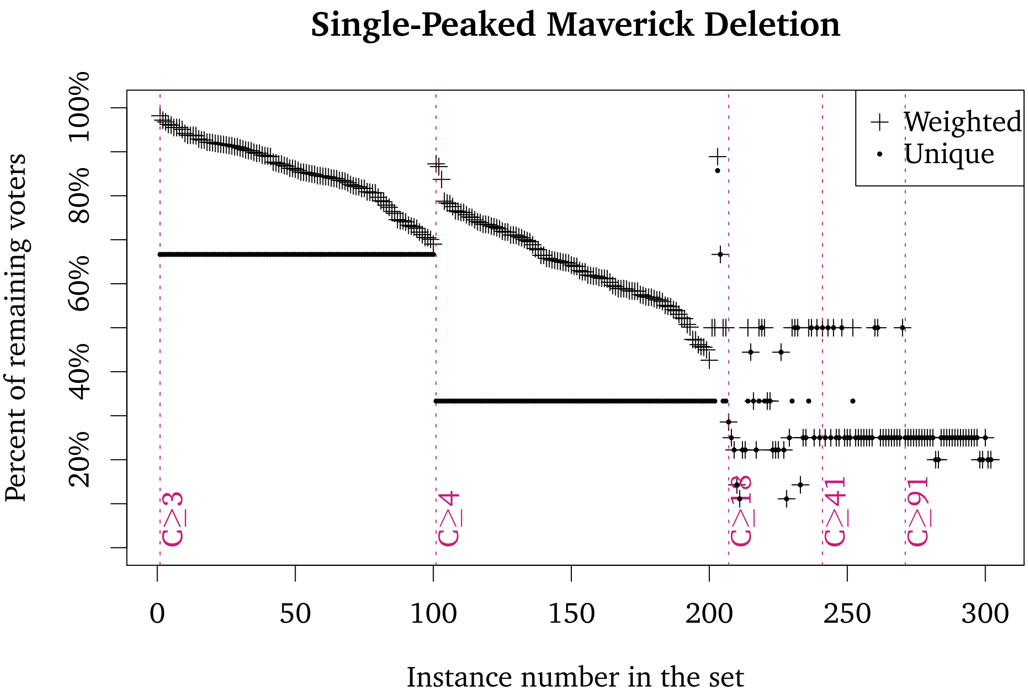

Fig. 5.17: Aggregated Results for Single-Peaked Maverick Deletion

As an interesting side-remark we also provide an aggregated performance comparison between single-peaked and single-crossing problems. Contrary to our expectations the performance across the whole set of instances is almost identical, albeit with more variance for maverick deletion (see Fig. 5.19 and Fig. 5.20).

Finally our assumption regarding maverick deletion on unique profiles being a lower bound for weighted profiles has proven true.

5.4 Future Work and Closing Remarks

Altogether we found very little evidence for the existence of nearly single-peaked or nearly single-crossing profiles in PrefLib. In our opinion only two of the eight data sets provided by PrefLib can be reasonably considered nearly single-peaked or nearly single-crossing . Furthermore there is no clear indication whether candidate or maverick deletion should yield better performance in practice.

Because of this uncertainty we would like to see further research especially on distances combining both candidate and maverick deletion. It should be noted,

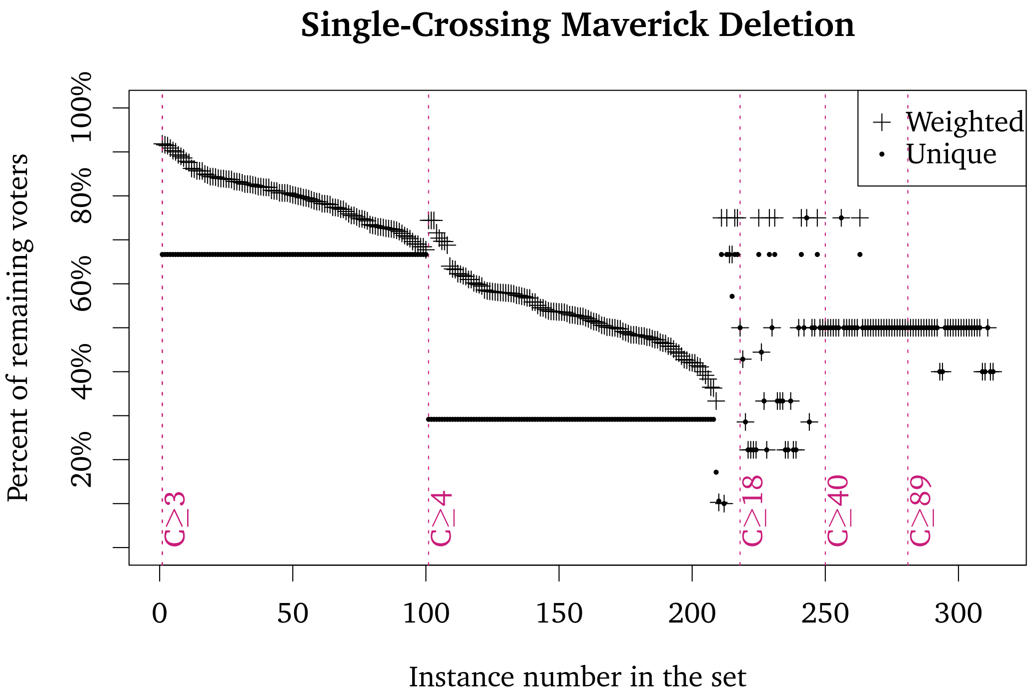

Fig. 5.18: Aggregated Results for Single-Crossing Maverick Deletion

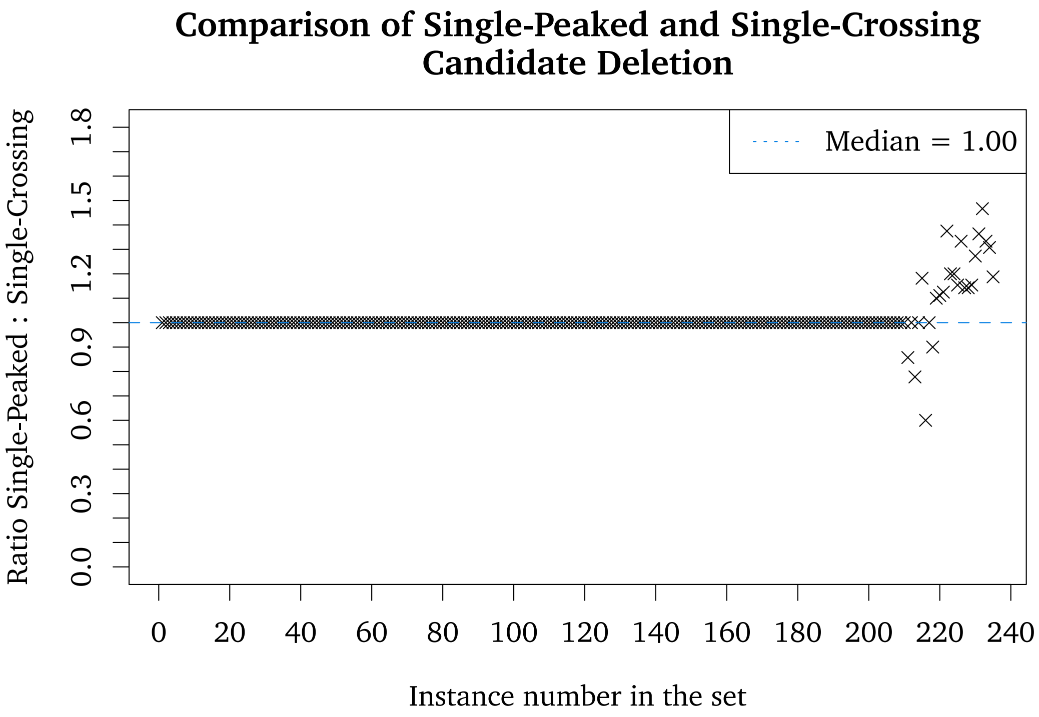

Fig. 5.19: Aggregated results for comparison of candidate deletion performance

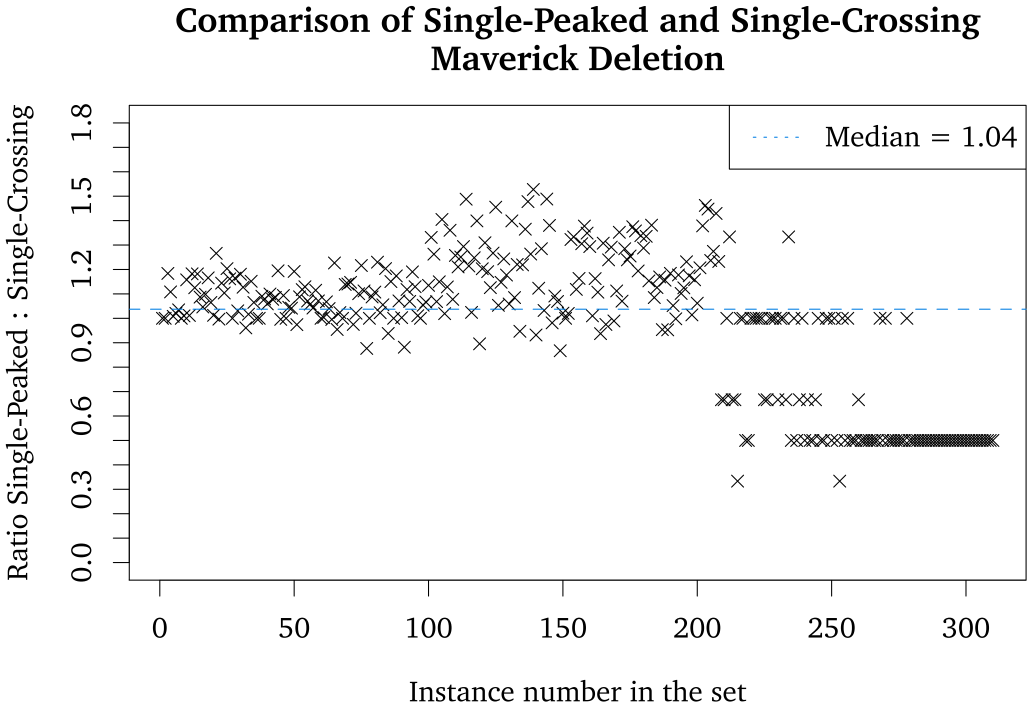

Fig. 5.20: Aggregated results for comparison of maverick deletion performance

however, that a suitable definition for such distance measure is a challenge in of itself.

It is possible that other distance measures, than the two we have employed, would have yield different results, and we would like to see such results if possible.

As we were unable to compute a result for Single-Crossing Candidate Deletion for about 25% of the instances, we hope that algorithmic improvements can eliminate a degree of uncertainty in our results.

Finally there are more types of restricted domains, albeit less popular, which could be employed to perform a similar analysis.

Bibliography

[Arr51] Kenneth J Arrow. “Social choice and individual values”. In: *Wiley, New York.* (1951) (cit. on p. 6).

[Asc78] K Michael Aschenbrenner. “Expected risk and single-peaked preferences”. In: *Acta Psychologica* 42.5 (1978), pp. 343–356 (cit. on p. 6).

[Asc81] K Michael Aschenbrenner. “Efficient sets, decision heuristics, and single-peaked preferences”. In: *Journal of Mathematical Psychology* 23.3 (1981), pp. 227–256 (cit. on p. 6).

[Avi95] Algirdas Avizienis. “The methodology of n-version programming”. In: *Software fault tolerance* 3 (1995), pp. 23–46 (cit. on p. 50).

[Bar+92] John J Bartholdi, Craig A Tovey, and Michael A Trick. “How hard is it to control an election?” In: *Mathematical and Computer Modelling* 16.8 (1992), pp. 27–40 (cit. on p. 7).

[BH07] Miguel A Ballester and Guillaume Haeringer. *A characterization of the single-peaked domain*. Tech. rep. Working paper, Universitat Autonoma de Barcelona, 2007 (cit. on p. 17).

[BI+89] John J Bartholdi III, Craig A Tovey, and Michael A Trick. “The computational difficulty of manipulating an election”. In: *Social Choice and Welfare* 6.3 (1989), pp. 227–241 (cit. on p. 7).

[BL76] Kellogg S Booth and George S Lueker. “Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms”. In: *Journal of Computer and System Sciences* 13.3 (1976), pp. 335–379 (cit. on pp. 21, 22).

[Bla+58] Duncan Black, Robert Albert Newing, Iain McLean, Alistair McMillan, and Burt L Monroe. *The theory of committees and elections*. Springer, 1958 (cit. on p. 6).

[Bla48] Duncan Black. “On the rationale of group decision-making”. In: *The Journal of Political Economy* (1948), pp. 23–34 (cit. on p. 6).

[Bra+15] Felix Brandt, Markus Brill, Edith Hemaspaandra, and Lane A Hemaspaandra. “Bypassing combinatorial protections: polynomial-time algorithms for single-peaked electorates”. In: *Journal of Artificial Intelligence Research* (2015), pp. 439–496 (cit. on p. 7).

[Bre+13a] Robert Bredereck, Jiehua Chen, and Gerhard J Woeginger. “A characterization of the single-crossing domain”. In: *Social Choice and Welfare* 41.4 (2013), pp. 989–998 (cit. on pp. 18, 21, 22, 51, 58).

[Bre+13b] Robert Bredereck, Jiehua Chen, and Gerhard J Woeginger. “Are there any nicely structured preference profiles nearby?” In: *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press. 2013, pp. 62–68 (cit. on pp. 21, 23, 52).

[BT86] John Bartholdi and Michael A Trick. “Stable matching with preferences derived from a psychological model”. In: *Operations Research Letters* 5.4 (1986), pp. 165–169 (cit. on pp. 16, 21, 49).

[CA77a] Clyde H Coombs and George Avrunin. “A theorem on single-peaked preference functions in one dimension”. In: *Journal of Mathematical Psychology* 16.3 (1977), pp. 261–266 (cit. on p. 6).

[CA77b] Clyde H Coombs and George S Avrunin. “Single-peaked functions and the theory of preference.” In: *Psychological review* 84.2 (1977), p. 216 (cit. on p. 6).

[CA78] Liming Chen and Algirdas Avizienis. “N-version programming: A fault-tolerance approach to reliability of software operation”. In: *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*. 1978, pp. 3–9 (cit. on p. 50).

[Che+10] Jianer Chen, Iyad A Kanj, and Ge Xia. “Improved upper bounds for vertex cover”. In: *Theoretical Computer Science* 411.40 (2010), pp. 3736–3756 (cit. on p. 54).

[Chi94] Stephen Ching. “An alternative characterization of the uniform rule”. In: *Social Choice and Welfare* 11.2 (1994), pp. 131–136 (cit. on p. 6).

[Coo69] Clyde H Coombs. “Portfolio theory: A theory of risky decision making”. In: *La decision* (1969) (cit. on p. 6).

[Cor+13] Denis Cornaz, Lucie Galand, and Olivier Spanjaard. “Kemeny elections with bounded single-peaked or single-crossing width”. In: *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press. 2013, pp. 76–82 (cit. on pp. 5, 7).

[Dem94] Gabrielle Demange. “Intermediate preferences and stable coalition structures”. In: *Journal of Mathematical Economics* 23.1 (1994), pp. 45–58 (cit. on p. 7).

[Dow57] Anthony Downs. “An economic theory of political action in a democracy”. In: *The journal of political economy* (1957), pp. 135–150 (cit. on p. 6).

[EL14] Edith Elkind and Martin Lackner. “On Detecting Nearly Structured Preference Profiles”. In: *Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014 (cit. on pp. 24, 54).

[Elk+12] Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. “Clone structures in voters’ preferences”. In: *Proceedings of the 13th ACM conference on electronic commerce*. ACM. 2012, pp. 496–513 (cit. on pp. 21, 22).

[Elk+15] Edith Elkind, Piotr Faliszewski, Martin Lackner, and Svetlana Obraztsova. “The complexity of recognizing incomplete single-crossing preferences”. In: *AAAI*. Vol. 15. 2015, pp. 865–871 (cit. on p. 12).

[EP98] Dennis Epple and Glenn J Platt. “Equilibrium and local redistribution in an urban economy when households differ in both preferences and incomes”. In: *Journal of Urban Economics* 43.1 (1998), pp. 23–51 (cit. on p. 7).

[Erd+13] Gábor Erdélyi, Martin Lackner, and Andreas Pfandler. “Computational Aspects of Nearly Single-Peaked Electorates”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013 (cit. on pp. 21, 23, 27, 46).

[Esc+08] Bruno Escoffier, Jérôme Lang, and Meltem Öztürk. “Single-Peaked consistency and its complexity”. In: *18th European Conference on Artificial Intelligence*. 2008, pp. 366–370 (cit. on pp. 21, 27, 29, 31, 35, 43, 45, 49, 58).

[Fal+06] Piotr Faliszewski, Edith Hemaspaandra, and Lane A Hemaspaandra. “The complexity of bribery in elections”. In: *Proceedings of the 21st national conference on Artificial intelligence-Volume 1*. AAAI Press. 2006, pp. 641–646 (cit. on p. 7).

[Fal+09] 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”. In: *Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge*. ACM. 2009, pp. 118–127 (cit. on p. 7).

[Fer10] Henning Fernau. “Parameterized algorithms for d-Hitting Set: The weighted case”. In: *Theoretical Computer Science* 411.16 (2010), pp. 1698–1713 (cit. on p. 54).

[Gra78] Jean-Michel Grandmont. “Intermediate preferences and the majority rule”. In: *Econometrica: Journal of the Econometric Society* (1978), pp. 317–330 (cit. on p. 7).

[GS96] Joshua S Gans and Michael Smart. “Majority voting with single-crossing preferences”. In: *Journal of Public Economics* 59.2 (1996), pp. 219–237 (cit. on p. 7).

[KUN06] FAN-CHIN KUNG. “An algorithm for stable and equitable coalition structures with public goods”. In: *Journal of Public Economic Theory* 8.3 (2006), pp. 345–355 (cit. on p. 7).

[Lac14] Martin Lackner. “Incomplete Preferences in Single-Peaked Electorates”. In: *Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014 (cit. on p. 12).

[Mat+12] Nicholas Mattei, James Forshee, and Judy Goldsmith. “An empirical study of voting rules and manipulation with large datasets”. In: *Proceedings of COMSOC* (2012) (cit. on p. 59).

[MF14] Krzysztof Magiera and Piotr Faliszewski. “How Hard is Control in Single-Crossing Elections”. In: *21st European Conference on Artificial Intelligence* (2014) (cit. on p. 7).

[MR81] Allan H Meltzer and Scott F Richard. “A rational theory of the size of government”. In: *The Journal of Political Economy* (1981), pp. 914–927 (cit. on p. 7).

[MW13] Nicholas Mattei and Toby Walsh. “PrefLib: A Library of Preference Data HTTP://PREFLIB.ORG”. In: *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*. Lecture Notes in Artificial Intelligence. Springer, 2013 (cit. on pp. 1, 7, 48).

[NK04] Rob Nederpelt and Fairouz Kamareddine. *Logical reasoning: a first course*. King’s College, 2004 (cit. on p. 14).

[Ott+96] Gert-Jan Otten, Hans Peters, and Oscar Volij. "Two characterizations of the uniform rule for division problems with single-peaked preferences". In: *Economic Theory* 7.2 (1996), pp. 291–306 (cit. on p. 6).

[Rob77] Kevin WS Roberts. "Voting over income tax schedules". In: *Journal of Public Economics* 8.3 (1977), pp. 329–340 (cit. on p. 7).

[Rot90] Paul Rothstein. "Order restricted preferences and majority rule". In: *Social Choice and Welfare* 7.4 (1990), pp. 331–342 (cit. on p. 7).

[San09] P. Sanders. "Algorithm engineering–an attempt at a definition". In: *Efficient Algorithms* (2009), pp. 321–340 (cit. on p. 47).

[Sap09] Alejandro Saporiti. "Strategy-proofness and single-crossing". In: *Theoretical Economics* 4.2 (2009), pp. 127–163 (cit. on p. 7).

[Spr91] Yves Sprumont. "The division problem with single-peaked preferences: a characterization of the uniform allocation rule". In: *Econometrica: Journal of the Econometric Society* (1991), pp. 509–519 (cit. on p. 6).

[ST06] Alejandro Saporiti and Fernando Tohmé. "Single-crossing, strategic voting and the median choice rule". In: *Social Choice and Welfare* 26.2 (2006), pp. 363–383 (cit. on p. 7).

[Tho94] William Thomson. "Consistent solutions to the problem of fair division when preferences are single-peaked". In: *Journal of Economic Theory* 63.2 (1994), pp. 219–245 (cit. on p. 6).

[Wah07] Magnus Wahlström. "Algorithms, measures and upper bounds for satisfiability and related problems". In: (2007) (cit. on p. 54).

[Wes77] Frank Westhoff. "Existence of equilibria in economies with a local public good". In: *Journal of economic Theory* 14.1 (1977), pp. 84–112 (cit. on p. 7).

List of Figures

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 4. Februar, 2016

Tomasz Przedmojski