FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Detecting Structure in Permutations and Preferences

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Martin Lackner
Registration Number 0525019

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Reinhard Pichler
Co-Advisor: Prof. Dr. Stefan Szeider

The dissertation has been reviewed by:

(Prof. Dr. Reinhard Pichler) (Prof. Dr. Gabór Erdélyi)

Vienna, 14th May 2014

(Martin Lackner)

Erklärung zur Verfassung der Arbeit

Martin Lackner
1080 Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Martin Lackner)

Dedicated to my parents and family

Acknowledgements

Science is a joint venture. So, first and foremost, I would like to thank my coauthors for the productive and enjoyable collaboration with them, the reviewers for their valuable comments, the conference organizers for great experiences all around the world, the many scientists I have met and talked to during my doctoral studies and, of course, the proverbial giants on whose shoulders I stand.

In particular, I would like to thank my advisor Reinhard Pichler for his tremendous support and for giving me the freedom to pursue my own goals, my colleagues Andreas Pfandler, Emanuel Sallinger, Martin Kronegger, Sebastian Skritek, Stefan Rümmele and Vadim Savenkov for their help and for making DBAI such a fun place to work at, my international collaborators Edith Elkind and Gabór Erdélyi for inviting me and for our successful collaboration, my co-advisor Stefan Szeider for helpful discussions and feedback, my colleagues from the doctoral program for the enjoyable student retreat in Payerbach, Katherine Tiede for her inspiring seminars on scientific writing, Johann Wruß for sparking my interest in mathematics and Marie-Louise Bruner for more than would fit on this page.

Finally, let me especially thank those who proofread parts of this thesis: Marie-Louise Bruner, Michael Gerstenecker, Andreas Pfandler, Reinhard Pichler and Sebastian Skritek.

This thesis was supported by the Austrian Science Fund (FWF): P25518-N23 and P20704-N18 and by the Vienna Science and Technology Fund (WWTF) project ICT12-15.

Abstract

The detection and subsequent utilization of structure in data is a major theme in algorithm design. While many algorithmic problems are computationally hard on arbitrary data, real-world data often possesses characteristics – structure – that allow to speed up computation. A necessary first step is to identify structure; for this task efficient algorithms are required. This thesis considers structure detection in two particular forms of data: permutations and preferences. Algorithmic, complexity theoretic and combinatorial methods are used with the aim of establishing tools for efficiently detecting structure.

Structure in permutations is studied in the form of permutation patterns. Detecting classical permutation patterns is NP-complete in general but requires only linear time for patterns of constant size. In this thesis, we explore the possibilities of detecting more general types of permutation patterns and show that these are considerably harder to detect than classical permutation patterns. For classical permutation patterns, we present a fast detection algorithm; the first to improve upon the exponential runtime of $\mathcal{O}^*(2^n)$, which is required by brute-force search.

Structure in preferences is studied in the form of domain restrictions. In computational social choice, domain restrictions are studied intensively as they often allow for efficient algorithms for otherwise intractable voting problems. Here, the detection of domain restrictions is a necessary precondition for their subsequent algorithmic utilization. This thesis considers the detection of structure in preferences from several viewpoints: First, we consider notions of distance to domain restrictions, which allow for more robust and flexible notions of structure. Although our results show that it is computationally hard to detect preferences which are only close to a domain restriction, we find efficient approximation and fixed-parameter algorithms solving this task. Second, we study single-peaked preferences (a particular form of domain restriction) in incomplete preferences. Here, depending on the exact notion of incompleteness, we find both intractable problems and fast algorithms.

Finally, we mathematically connect permutation patterns with domain restrictions and thus establish a link between the two main concepts in this thesis. This link allows us to use methods from permutation patterns to identify combinatorial properties of domain restrictions. These results are the first to make precise statements about the likelihood of domain restrictions in random preferences. Also, we use this link to establish limits for the efficient detection of domain restrictions.

Kurzfassung

Die Erkennung und anschließende Nutzung von Struktur in Daten ist ein Kerngebiet des Algorithmenentwicklung. Während viele algorithmische Probleme im Allgemeinen rechenaufwändig sind, erlaubt die Strukturiertheit von Daten aus der realen Welt eine Beschleunigung der Rechenvorgänge. Ein notwendiger erster Schritt ist die Erkennung von Struktur, wofür effiziente Algorithmen erforderlich sind. Diese Dissertation beschäftigt sich mit der Erkennung von Struktur in zwei besonderen Arten von Daten: Permutationen und Präferenzen. Es werden Algorithmen, komplexitätstheoretische und kombinatorische Methoden verwendet mit dem Ziel, effiziente Mittel zur Strukturerkennung zur Verfügung zu stellen.

Struktur in Permutationen wird in Form von Permutationsmustern untersucht. Das Erkennen von klassischen Permutationsmustern ist im Allgemeinen NP-vollständig, braucht aber nur lineare Zeit für Muster konstanter Länge. In dieser Arbeit erforschen wir die Möglichkeiten, noch allgemeinere Permutationsmuster zu erkennen, und zeigen, dass diese bedeutend schwieriger zu entdecken sind als klassische Muster. Für klassische Permutationsmuster präsentieren wir einen schnellen Erkennungsalgorithmus. Dies ist der erste Algorithmus, der die exponentielle Laufzeit von $\mathcal{O}^*(2^n)$ unterbietet, die bei einer Brute-Force-Suche benötigt wird.

Struktur in Präferenzen wird in Form von Domain-Restrictions untersucht. In Computational Social Choice werden Domain-Restrictions intensiv untersucht, da sie oft Algorithmen für Wahlprobleme ermöglichen, die andernfalls sehr aufwändig zu berechnen wären. Hier ist die Erkennung von Domain-Restrictions ein notwendiger erster Schritt für ihre darauffolgende algorithmische Nutzung. Diese Dissertation untersucht Struktur in Präferenzen aus verschiedenen Blickrichtungen: Erstens werden verschiedene Arten von Distanz zu Domain-Restrictions untersucht. Die Distanzmaße erlauben robustere und flexiblere Definitionen von Struktur in Präferenzen. Obwohl unsere Resultate zeigen, dass es sehr rechenintensiv ist zu überprüfen, ob Präferenzen nahe einer Domain-Restriction sind, finden wir effiziente Approximations- und parametrisierte Algorithmen, die diese Aufgaben lösen. Zweitens untersuchen wir single-peaked Präferenzen (dies ist eine bestimmte Form von Domain-Restriction) in unvollständigen Präferenzen. Hier finden wir, abhängig von der genauen Definition von Unvollständigkeit, sowohl schwer zu berechnende Probleme als auch effiziente Algorithmen.

Abschließend verbinden wir Permutationsmuster mit Domain-Restrictions und schaffen hiermit eine Brücke zwischen den beiden großen Themenkomplexen dieser Dissertation. Diese Verbindung erlaubt, Methoden aus dem Gebiet der Permutationsmuster zu verwenden, um kombinatorische Eigenschaften von Domain-Restrictions zu finden. Diese Resultate erlauben, dass erstmals präzise Aussagen über die Häufigkeit von Domain-Restrictions in vollständig zufälligen

Präferenzen getroffen werden. Darüber hinaus verwenden wir diese Verbindung, um auszuloten, bis zu welchem Grad eine effiziente Erkennung von Domain-Restrictions möglich ist.

Contents

CHAPTER 1

Introduction

1.1 Detection of Structure

Making large quantities of information accessible is a central goal of computer science. This goal is at the core of many disciplines of computer science such as data visualization, knowledge representation, database architecture. For both humans and machines it is challenging to handle large amounts of data. Interestingly, both humans and computers can handle large quantities of data with the same basic strategy: by detecting and utilizing its structure.

This thesis deals with structure detection in two particular forms of data: permutations, i.e., orderings of natural numbers, and preferences, i.e., rankings of options. Both permutations and preferences are two very general forms of data and appear in a wide range of applications. Permutations are a fundamental object in mathematics and appear in almost any mathematical area. Permutations also appear in applications such as error detection codes or mathematical biology. Preferences appear in a wide range of sciences and applications ranging from artificial intelligence and (computational) social choice to economy. While these two topics, permutations and preferences, are seemingly unrelated, this thesis establishes a close connection between structure in permutations and in preferences.

In this thesis, we consider a particular form of structure in permutations: *permutation patterns*. A permutation T contains a permutation P as a pattern if there exists a subsequence of T that has the same relative order of elements as P. For example 53142 contains 231 as shown by the subsequence 342 (cf. Figure 1.1). On the other hand 53142 avoids 123 since it does not contain an increasing subsequence of length 3. Permutation patterns are an extensively studied topic with applications in areas such as bioinformatics (genome rearrangement) [36, 38, 50] and sorting algorithms [31, 141]. This concept of (classical) permutation patterns has been expanded to generalized permutation patterns, where additional constraints have to be satisfied by pattern occurrences. (For an overview of generalizations of permutation patterns, the reader is referred to Chapter 4.1.)

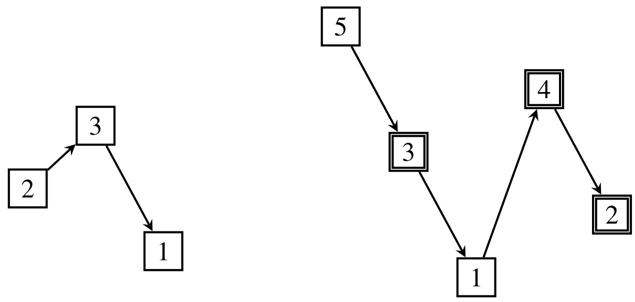


Figure 1.1: The pattern 231 (left-hand side) is contained in the permutation 53142 (right-hand side).

While both classical and generalized permutation patterns have been studied extensively from a combinatorial point of view, far less is known about the computational aspects of detecting permutation patterns. In particular, the following questions have not yet been answered.

Generalized Patterns. The problem of detecting permutation patterns with variable length is known to be NP-complete in general [34]. However, Guillemot and Marx [92] showed that patterns of constant length can be found in linear time. It is not clear whether this result can be extended to generalized permutation patterns. More generally, what is the complexity of detecting generalized permutation patterns and is it possible to find short patterns efficiently?

Fast Detection. Is there a fast algorithm for finding classical permutation patterns if one cannot assume the pattern to be short? More specifically, the trivial brute-force algorithm has a runtime of $\mathcal{O}(2^n \cdot n)$. So far, no algorithm has been discovered that improves the exponential runtime to c^n for some constant $c < 2$. Is this possible?

Structure in preferences has been studied mainly in the form of *domain restrictions*, for example the single-peaked restriction. For an intuitive understanding of single-peakedness consider the following situation. A group of coworkers wants to decide upon the temperature in a work space. The options are 18, 19, 20, 21 and 22 degrees Celsius. The preferences of each worker correspond to a ranking of these options. Some possible rankings do not seem to be plausible in this scenario. For example if a worker prefers 19 degrees, her second ranked option will not be 22 degrees but rather 20 or 18 degrees. In general one can assume that if each worker has a preferred option (the peak), the options left and right on the temperature axis will be ranked in a decreasing order. See Figure 1.2 for an example of two single-peaked preferences.

Domain restrictions are of particular interest for algorithmic purposes: computationally hard problems concerned with preference data are often solvable by fast algorithms if a domain restriction can be assumed. It is therefore of considerable interest to design algorithms that detect domain restrictions. Such algorithms are usually a necessary prerequisite to apply algorithms tailored to a specific restricted domain.

In this thesis we explore domain restrictions and more generally the structure of preferences. Several algorithmic problems regarding the detection of structure in preferences have not been tackled so far:

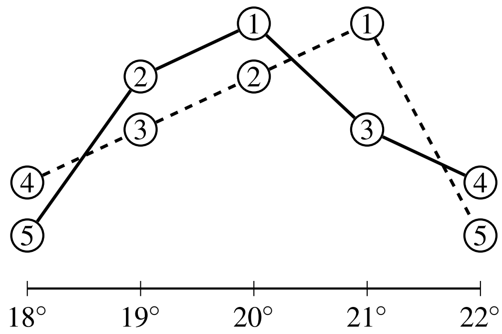


Figure 1.2: Two preferences that are single-peaked with respect to the temperature axis

Nearly Structured Preferences. Preferences often do possess some kind of structure. However, as experiments have shown, domain restrictions such as the single-peaked restriction are too restrictive for real data. As a remedy, some notions of distance to single-peakedness have been proposed to be able to speak about "closeness" to a domain restriction. What is the relation of these notions of distance and is it possible to efficiently verify whether preferences are close to a domain restriction?

Incomplete Preferences. A common assumption in social choice theory is that preferences are given as total orders. Does the computational complexity of structure detection vary if one assumes more realistic models for preference data? So far, structure in incomplete preferences has not been considered in the literature.

Connections. The main difference between preferences and permutations is that natural numbers have an underlying order whereas arbitrary options do not have such an order. As soon as an order is established on the options – as it is the case for single-peaked preferences – a visual similarity between preferences and permutations appears (cf. the up-and-down visualization in Figure 1.1 and Figure 1.2). Is there a deeper connection between structure in permutations (permutation patterns) and structure in preferences (domain restrictions)? If so, can this connection be used to transfer results from one domain to the other?

1.2 Goal and Main Results

The overarching aim of this thesis is to provide means to detect structure in permutations and preferences. Towards this aim, we use algorithmic, complexity theoretic and combinatorial methods.

Part I of this thesis deals with permutation patterns. The results of this part can be summarized as follows:

Generalized Patterns. We show for generalizations of classical permutation patterns that no fixed-parameter algorithm exists under the common complexity theoretic assumption $\mathsf{FPT} \neq$

W[1]. However, for two types of patterns that are restricted forms of generalized patterns, we find polynomial-time algorithms. These results can be found in Chapter 4.

Fast Detection. For classical patterns, we present a fixed-parameter algorithm for permutation pattern matching with a worst-case runtime of $\mathcal{O}(1.79^{\mathsf{run}(T)} \cdot n \cdot k)$, where $\mathsf{run}(T)$ denotes the number of alternating runs of T. Alternating runs describe the up-and-down structure of permutations; for example, the permutation 53142 consists of three runs (cf. Figure 1.1). Since $\mathsf{run}(T) < n$, this yields a $\mathcal{O}(1.79^n \cdot n \cdot k)$ algorithm. Thus, this is the first algorithm that improves upon the $\mathcal{O}^*(2^n)$ runtime required by brute-force search without imposing restrictions on P and T. Furthermore we prove that – under standard complexity theoretic assumptions – such a fixed-parameter tractability result is not possible for $\mathsf{run}(P)$. These results can be found in Chapter 5.

Part II of this thesis deals with structure in preferences:

Nearly Structured Preferences. We introduce several new distance measures regarding single-peakedness. We prove that determining whether a given profile is nearly single-peaked is NP-complete in many cases. For one case (deleting options to achieve single-peakedness) we present a polynomial-time algorithm. We also explore the relations between these notions of nearly single-peakedness. These results can be found in Chapter 6.

For those problems that turn out to be NP-hard, we develop efficient approximation algorithms. Our algorithms are not only applicable to the single-peaked restriction but to all domains that can be characterized in terms of forbidden configurations. For a large range of scenarios, our approximation results are optimal under a plausible complexity-theoretic assumption. We also provide parameterized complexity results for this class of problems. All these results can be found in Chapter 7.

Incomplete Preferences. While checking single-peakedness for complete preferences can be done in linear time, this problem is NP-complete for incomplete preferences. Despite this computational hardness result, we find four polynomial-time algorithms for reasonably restricted settings. These results can be found in Chapter 8.

Connections. We establish a close connection between the two main objects that are studied in this thesis: permutation patterns and domain restrictions. This connection is used to apply results from permutation patterns to the field of domain restrictions. Two main tasks are accomplished due to this connection. First, we perform a combinatorial analysis of domain restrictions. Our results indicate that it is very unlikely that random preference profiles belong to a restricted domain. Second, we analyze the computational complexity of detecting domain restrictions. These complexity results are obtained through reduction from permutation pattern detection problems. This connection and the corresponding results can be found in Chapter 9.

For a more detailed account of the results obtained in this thesis, we refer the reader to the summaries at the end of each chapter.

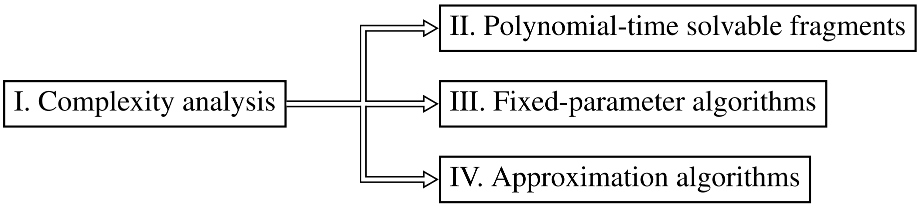


Figure 1.3: The sequence of methods used in our algorithmic analysis

1.3 Methodology

This thesis takes mostly an algorithmic point of view. The aim is to find efficient algorithms for precisely defined computational problems. This algorithmic approach is complemented by a computational complexity analysis wherein complexity results allow us to prove impossibility results for efficient algorithms, usually under some complexity theoretic assumption. We distinguish four main parts in our algorithmic analysis (cf. Figure 1.3).

I. First, we perform a complexity analysis of the problem at hand. This might be a classical complexity analysis or a parameterized complexity analysis. A classical complexity analysis allows ruling out polynomial-time algorithms (assuming $\mathsf{P} \neq \mathsf{NP}$). A parameterized complexity analysis permits ruling out fixed-parameter algorithms (assuming $\mathsf{FPT} \neq \mathsf{W[1]}$). The complexity classes $\mathsf{P}, \mathsf{NP}, \mathsf{FPT}, \mathsf{W[1]}$ and others are explained in Chapter 2. In this thesis, this first step generally yields intractability results, in particular NP-completeness results. Thus, further algorithmic techniques are required to achieve our aim of efficient algorithms. The following three techniques used in our algorithmic analysis are applicable to computationally hard problems.

II. NP-hardness does not rule out the possibility of polynomial-time algorithms for a smaller problem domain, that is, obtaining polynomial-time algorithms by restricting the set of allowable instances. A classical example of this approach is the linear-time algorithm for 2-SAT [11], restricting the SATISFIABILITY problem to instances with clauses of size 2. For instances of size 3, i.e., 3-SAT, the problem remains NP-hard. Similar to this example, we try to meaningfully restrict our problems so that they become polynomial-time solvable.

III. Another approach to deal with computational hardness is to identify parameters that make a problem instance computationally demanding. As a typical example let us consider the NP-complete VERTEX COVER problem. This problem asks, given a graph, for a minimum subset of vertices such that every edge has an endpoint in that subset. A possible parameter is the size of the solution, i.e, a value k such that a vertex cover of size k exists. VERTEX COVER can be solved in time $\mathcal{O}(2^k \cdot n)$. The runtime of this algorithm thus depends exponentially on k but only polynomially (even linearly) on the input size. Such an algorithm is called fixed-parameter tractable with respect to the parameter k. Fixed-parameter (tractable) algorithms can be applied to arbitrary problem instances (in contrast

		I.	II.	III.	IV.
Permutation patterns	Chapter 4	✓	✓		NA
	Chapter 5			✓	
Preferences	Chapter 6	✓	✓		
	Chapter 7			✓	✓
	Chapter 8	✓	✓		
	Chapter 9	✓			

Figure 1.4: Which techniques (cf. Figure 1.3) are used in which chapter

to the previous approach) but are only fast if the corresponding parameter is small. Since real-world instances usually possess some kind of structure, it is reasonable to assume that usually some parameter is small. The search for useful parameters and corresponding fast fixed-parameter algorithms is thus a viable approach to dealing with computational hardness.

IV. In contrast to exact fixed-parameter algorithms, approximation algorithms are sometimes acceptable although they only yield approximate solutions. Approximation algorithms – with guaranteed approximation accuracy – may require only polynomial time even for NP-hard problems. A classical example is VERTEX COVER: computing a vertex cover that is at most twice as large as the optimal solution can be done in linear time [72]. Analogously, we want to find approximation algorithms for otherwise hard structure detection problems. In Chapter 7 we will see that approximation algorithms are very well suited for efficiently detecting nearly structured preferences.

We refer the reader to Figure 1.4 that indicates which technique has been applied in which chapter of this thesis. Chapter 9, in addition to complexity results, also uses combinatorial methods to make statements about the likelihood of structure. We would like to remark that the framework of approximation algorithms is not applicable (NA) to decision problems. Finding permutation patterns is a typical example for such a problem, since either there is a matching or not – there is no intermediate outcome possible. Thus, Chapter 4 and 5, the chapters concerning permutation patterns, do not deal with approximation algorithms. To make approximation algorithms applicable, one would first have to define a corresponding optimization problem, e.g., to ask what is the largest subsequence of the pattern such that this subsequence can be matched. However, generalizations of that sort are not considered in this thesis.

Finally, we would like to state that this thesis is a theoretical treatment of the questions at hand. While the algorithms presented here are applicable to real-life data sets, experiments are not part of the thesis. In Chapter 10 we discuss implementations and experiments as a future research direction.

1.4 Publications

This thesis is based on the following publications:

- Marie-Louise Bruner and Martin Lackner. A fast algorithm for permutation pattern matching based on alternating runs. In *Proceedings of the 13th Scandinavian Workshop on Algorithm Theory (SWAT 2012)*, volume 7357 of *Lecture Notes in Computer Science*, pages 261–270. Springer, 2012.

- Marie-Louise Bruner and Martin Lackner. The computational landscape of permutation patterns. *Pure and Applied Mathematics*, 2014. Accepted for publication.

- Gabór Erdélyi, Martin Lackner, and Andreas Pfandler. Computational aspects of nearly single-peaked electorates. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2013)*, 2013.

- Martin Lackner. Incomplete preferences in single-peaked electorates. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

- Edith Elkind and Martin Lackner. On detecting nearly structured preference profiles. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

- Marie-Louise Bruner and Martin Lackner. The likelihood of structure in preference profiles. In *Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPref 2014)*, 2014. Accepted for publication.

The following publications have been obtained by the thesis author during his doctoral studies but are not part of this thesis:

- Martin Kronegger, Martin Lackner, Andreas Pfandler and Reinhard Pichler. A parameterized complexity analysis of generalized CP-nets. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

- Martin Lackner and Andreas Pfandler. Fixed-parameter algorithms for finding minimal models. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 85–95. AAAI Press, 2012.

- Martin Lackner and Andreas Pfandler. Fixed-parameter algorithms for closed world reasoning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 492–497. IOS Press, 2012.

- Martin Lackner, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut on graphs of bounded clique-width. In *Proceedings of the 6th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2012)*, volume 7402 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2012.

CHAPTER 2

Preliminaries

The aim of this chapter is to introduce the basic concepts and notations that are used throughout this thesis.

2.1 Sets, Orders, Permutations

For any $m, n \in \mathbb{N}$ with $m \leq n$, let $[m, n]$ denote the set $\{m, m+1, \ldots, n\}$ and $[n]$ the set $\{1, 2, \ldots, n\}$.

2.1.1 Orders

Let S be a finite set. A *partial order* of S is a binary relation that is reflexive, antisymmetric and transitive. A *total order* of S is a partial order that is total, i.e., for every $a, b \in S$, either the pair (a, b) or (b, a) is contained in the relation. Let P be a partial order of S. Instead of writing $(a, b) \in P$, we write $a \leq_P b$ or $b \geq_P a$. We write $a <_P b$ or $b >_P a$ to state that $a \leq_P b$ and $a \neq b$. Sometimes, if the considered order is clear from the context, we omit the index of $>, \geq, <, \leq$, etc. Given two subsets A and B of S, we write $A >_P B$ to denote that every element in A is larger than every element in B with respect to P.

Let T be a total order on S. We write $T(i)$ to denote the i-th largest element with respect to T. A total order T is a *linearization* of a partial order P if $dom(T) = dom(P)$ and for all $a, b \in dom(P)$, $a <_P b$ implies $a <_T b$.

2.1.2 Permutations

A permutation is a bijective function from a finite set onto itself. An m-permutation is a permutation from $[m]$ to $[m]$. An m-permutation π can be seen as the sequence $\pi(1), \pi(2), \ldots, \pi(m)$. Viewing permutations as sequences allows us to speak of *subsequences* of a permutation. We speak of a *contiguous subsequence* of π if the sequence consists of contiguous elements in the sequence corresponding to π. Given a set $S \subseteq [m]$, we write $\pi|_S$ to denote the subsequence of π consisting exactly of the elements of S.

We denote by π^{-1} the inverse of the permutation π, by $\pi^r := \pi(n)\pi(n-1)\ldots\pi(1)$ its reverse and by $\pi^c := (n-\pi(1)+1)(n-\pi(2)+1)\ldots(n-\pi(n)+1)$ its complement.

Every $[m]$-permutation π defines a total order $\prec_\pi$ on $[m]$. We write $i \prec_\pi j$ if $\pi^{-1}(i) < \pi^{-1}(j)$, i.e., the value i stands to the left of the value j in π. We say i is left (right) of j if either $i \prec_\pi j$ ($j \prec_\pi i$) or $i = j$. We say i is strictly left (right) of j if i is left (right) of j and $i \neq j$.

2.1.3 Valleys, Peaks and Runs in Permutations

We discern two types of local extrema in permutations: valleys and peaks. A *valley* of a permutation π is an element $\pi(i)$ for which it holds that $\pi(i-1) > \pi(i)$ and $\pi(i) < \pi(i+1)$. If $\pi(i-1)$ or $\pi(i+1)$ is not defined, we still speak of valleys. Similarly, a *peak* denotes an element $\pi(i)$ for which it holds that $\pi(i-1) < \pi(i)$ and $\pi(i) > \pi(i+1)$.

Valleys and peaks partition a permutation into contiguous monotone subsequences, so-called *(alternating) runs*. The first run of a given permutation starts with its first element (which is also the first local extremum) and ends with the second local extremum. The second run starts with the following element and ends with the third local extremum. Continuing in this way, every element of the permutation belongs to exactly one alternating run. Observe that every alternating run is either increasing or decreasing. We therefore distinguish between *runs up* and *runs down*. Note that runs up always end with peaks and runs down always end with valleys. The parameter $\mathsf{run}(\pi)$ counts the number of alternating runs in π. Hence, $\mathsf{run}(\pi)+1$ equals the number of local extrema in π. These definitions can be analogously extended to subsequences of permutations.

Example 2.1. In the permutation $1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$ the valleys are 1, 4, 2 and 5 and the peaks are 12, 11, 9 and 10. A decomposition into alternating runs is given by:

$$1\,8\,12|4|7\,11|6\,3\,2|9|5|10.$$

For a graphical representation of this permutation the reader is referred to Figure 2.1. $\dashv$

2.1.4 Permutation Patterns

Definition 2.1. *Let P (the pattern) be a k-permutation. We say that an n-permutation T (the text)* contains P as a pattern *or that P* can be matched into T *if we can find a subsequence of T that is order-isomorphic to P. Matching P into T thus consists in finding a monotonically increasing function $M : [k] \to [n]$ so that the sequence $M(P)$, defined as*

$$\big(M(P(1)), M(P(2)), \ldots, M(P(k))\big),$$

is a subsequence of T. Such a function M is called a matching.

Example 2.2. Let us consider the text permutation $1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$ and the pattern permutation $2\,3\,1\,4$. A graphical representation can be found in Figure 2.1. Observe that the pattern can be matched into the text as witnessed by the subsequence $4\,6\,2\,9$. $\dashv$

For an extensive mathematical treatment of permutation patterns the reader is referred to Bóna's *Combinatorics of permutations* [32].

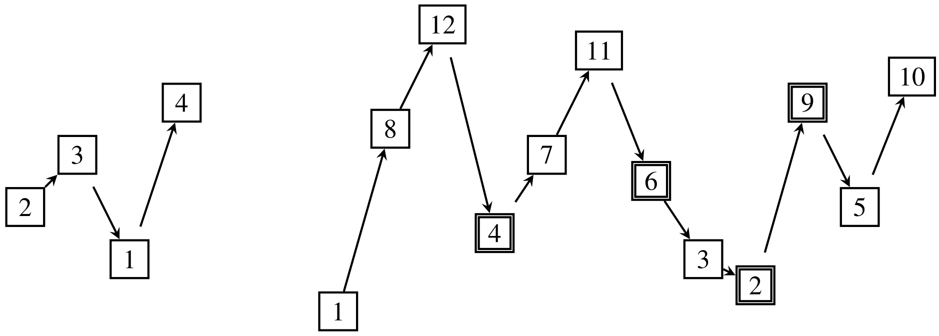


Figure 2.1: The pattern $2\,3\,1\,4$ (left-hand side) is contained in the text permutation $1\,8\,12\,4\,7\,11\;\,6\,3\,2\,9\,5\,10$ (right-hand side).

2.2 Preferences and Social Choice

2.2.1 Preferences and Elections

An *election* E is described by a set of *candidates* $C = \{c_1, \ldots, c_m\}$ and an ordered list of *votes* $\mathcal{P} = (V_1, \ldots, V_n)$. Each vote V_i, $i \in [n]$, is a total order over C. We refer to V_i as the vote, or *preferences*, of voter i, and write $E = (C, \mathcal{P})$. The list of votes $\mathcal{P}$ is called the *preference profile*, or *profile* for short.

For a vote V_i, we use $x \succ_i y$ to denote that x is larger than y with respect to the total order V_i, i.e., $(yV_ix) \wedge (x \neq y)$. As a shorthand notation we sometimes write $V_i : abc$ to denote that vote V_i is the total order $a \succ_i b \succ_i c$. If there is only one vote under consideration, usually denoted by V, we omit the index and write $x \succ y$. To easier distinguish between votes and other orders, we use the symbol $\succ$ to compare candidates with respect to a vote and $>$ for other orders.

Given a profile $\mathcal{P}'$, we write $\mathcal{P}' \subseteq \mathcal{P}$ if $\mathcal{P}'$ can be obtained from $\mathcal{P}$ by deleting some of the votes. Further, given $\mathcal{P}' \subseteq \mathcal{P}$, we write $\mathcal{P} \setminus \mathcal{P}'$ to denote the profile that can be obtained from $\mathcal{P}$ by removing the votes in $\mathcal{P}'$.

Given a vote V and a set of candidates $C' \subseteq C$, we define $V[C']$ to be the vote V restricted to candidates in C'. More generally, given a total order T with domain S and $S' \subseteq S$, we use $T[S']$ to denote the total order T restricted to elements in S'. Analogously, given a preference profile $\mathcal{P} = (V_1, \ldots, V_n)$, we define $\mathcal{P}[C']$ to be the restricted profile $(V_1[C'], \ldots, V_n[C'])$.

Unless explicitly stated otherwise, we denote the number of candidates with m and the number of votes with n.

Given a vote $V_i : c_1c_2 \ldots c_m$, let the vote $\overline{V_i} : c_mc_{m-1} \ldots c_1$ denote the *reverse* vote of V_i. More generally, the reverse of a total order T is denoted by $\overline{T}$.

2.2.2 Domain Restrictions

In what follows, we discuss restricted preference domains, i.e., sets of elections that satisfy certain properties. The single-peaked restriction [30] is the most widely used restriction. It assumes that the candidates can be ordered linearly on the so-called axis and voters prefer candidates close to their ideal point over candidates that are further away. For an example of single-peaked preferences, the reader is referred to the introduction and in particular to Figure 1.2. Throughout this thesis, let $(C, \mathcal{P})$ be an election.

Definition 2.2. *Let A be a total order of C, the so-called* axis. *A vote $V \in \mathcal{P}$* contains a valley *with respect to an axis A on the candidates $c_1, c_2, c_3 \in C$ if $c_1 <_A c_2 <_A c_3$, $c_2 \prec_V c_1$ and $c_2 \prec_V c_3$ holds. The profile $\mathcal{P}$ is* single-peaked with respect to A *if for every $V \in \mathcal{P}$ and for all candidates $c_1, c_2, c_3 \in C$, V does not contain a valley with respect to A on c_1, c_2, c_3. The profile $\mathcal{P}$ is* single-peaked consistent *(or simply,* single-peaked*) if there exists a total order A of C such that $\mathcal{P}$ is single-peaked with respect to A.*

The single-peaked restriction can be relaxed to a two-dimensional setting [17], in which valleys are less likely to arise. The intuition behind 2D single-peaked preferences is that there is an ideal point in the two-dimensional space and, again, candidates that are closer to this point are more preferred.

Definition 2.3. *Let A and B be total orders of C, the so-called axes. A vote $V \in \mathcal{P}$* contains a 2D-valley *with respect to (A, B) on the candidates $c_1, c_2, c_3 \in C$ if V contains a (1D) valley with respect to A on c_1, c_2, c_3 as well as a valley with respect to B on c_1, c_2, c_3. The profile $\mathcal{P}$ is* 2D single-peaked with respect to (A, B) *if for every vote $V \in \mathcal{P}$ and for all candidates $c_1, c_2, c_3 \in C$, V does not contain a 2D-valley with respect to (A, B) on c_1, c_2, c_3. The profile $\mathcal{P}$ is* 2D single-peaked *if there exist two total orders A, B of C such that $\mathcal{P}$ is 2D single-peaked with respect to (A, B).*

We continue with the single-crossing restriction [125], where the votes and not the candidates are ordered along a linear axis.

Definition 2.4. *Let A be a total order of $[n]$. The profile $\mathcal{P} = (V_1, \ldots, V_n)$ is* single-crossing with respect to A *if for every pair of candidates $c_1, c_2 \in C$ the set $\{i \in [n] \mid c_1 \prec_i c_2\}$ is an interval with respect to A. The profile $\mathcal{P}$ is* single-crossing *if there exists a total order A of $[n]$ such that $\mathcal{P}$ is single-crossing with respect to A.*

Note that both $\{i \in [n] \mid c_1 \prec_i c_2\}$ and $\{i \in [n] \mid c_2 \prec_i c_1\}$ have to form an interval with respect to the total order A. Thus, the (indices of) voters that prefer c_1 over c_2 precede the (indices of) voters that prefer c_2 over c_1 on A – or vice versa.

Further domain restrictions, such as the worst-/best-/medium-/value-restricted, single-caved and group-separable restriction, are defined in Section 7.1.

2.3 Algorithms and Computational Complexity

2.3.1 Classical Complexity Theory

We give a brief reminder of the two fundamental classes P and NP. The class P contains all problems that can be solved in polynomial time on a deterministic Turing machine. It is important to note that polynomial time means polynomial in the size of the input. The class NP contains all problems that can be solved in polynomial time on a *non-deterministic* Turing machine. A problem is NP-hard if every problem in NP can be reduced to it by a polynomial time reduction. Furthermore, a problem is NP-complete if it is contained in NP and NP-hard. For a detailed introduction to complexity theory the reader is referred to the monographs by Papadimitriou [122], Goldreich [90], and Arora and Barak [6].

2.3.2 Parameterized Complexity Theory

We give the relevant definitions of parameterized complexity theory. In contrast to classical complexity theory, a parameterized complexity analysis studies the runtime of an algorithm with respect to an additional parameter and not just the input size $|I|$. Therefore, every parameterized problem is considered as a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet. An instance of a parameterized problem consequently consists of an input string together with a positive integer p, the parameter.

Definition 2.5. *A parameterized problem is* fixed-parameter tractable *(or in* FPT*) if there is a computable function f such that there exists an algorithm solving an instance (I, k) in time $\mathcal{O}(f(k) \cdot |I|^{\mathcal{O}(1)})$.*

The algorithm itself is also called fixed-parameter tractable (fpt). When discussing fpt algorithms, we use the standard notation of parameterized complexity and write $\mathcal{O}^*(f(k))$ as a shorthand for $\mathcal{O}(f(k) \cdot |I|^{\mathcal{O}(1)})$, i.e., the $\mathcal{O}^*$ notation ignores polynomial factors.

A central concept in parameterized complexity theory are *fixed-parameter tractable reductions*, which allow for a parameterized hardness theory.

Definition 2.6. *Let $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. An* fpt-reduction *from L_1 to L_2 is a mapping $R : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ such that*

- *$(I, k) \in L_1$ if and only if $R(I, k) \in L_2$.*
- *R is computable by an fpt-algorithm.*
- *There is a computable function g such that for $R(I, k) = (I', k')$, $k' \leq g(k)$ holds.*

Besides the class FPT, other important complexity classes in the framework of parameterized complexity are W[1] $\subseteq$ W[2] $\subseteq \ldots$, building the so-called W-hierarchy. For our purpose, only the class W[1] is relevant. It is conjectured (and widely believed) that W[1] $\neq$ FPT. Therefore, showing W[1]-hardness can be considered as evidence that the problem is not fixed-parameter tractable.

Definition 2.7. *The class* W[1] *is defined as the class of all problems that are fpt-reducible to the following problem.*

TURING MACHINE ACCEPTANCE	
Instance:	A nondeterministic Turing machine with its transition table, an input word x and a positive integer k.
Parameter:	k
Question:	Does the Turing machine accept the input x in at most k steps?

Definition 2.8. *A parameterized problem is in* XP *if it can be solved in time* $\mathcal{O}(|I|^{f(k)})$ *where* f *is a computable function.*

All the aforementioned classes are closed under fpt-reductions. The following relations between these complexity classes are known:

$$\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \ldots \subseteq \mathsf{XP}$$

$$\mathsf{FPT} \subset \mathsf{XP}.$$

Further details can be found, for example, in the monographs by Downey and Fellows [63, 64], Niedermeier [120] and Flum and Grohe [80].

2.3.3 Approximation algorithms

In this thesis, we require only informal definitions of approximation algorithms; in particular, we only consider minimization problems. Given an instance I of an optimization problem, let $\mathsf{opt}(I)$ denote the size of a minimum (optimal) solution of I. In this thesis, only a specific type of approximation algorithm appears: constant-factor approximation. Let $c > 1$ be a constant. A c-approximation algorithm is a polynomial-time algorithm that solves an instance I of an optimization problem by returning a solution of size at most $c \cdot \mathsf{opt}(I)$.

CHAPTER 3

Related Work

3.1 Permutation Pattern Matching

The concept of pattern matching in permutations arose in the late 1960ies. It was in an exercise of his *Fundamental algorithms* [109] that Knuth asked which permutations could be sorted using a single stack. The answer is simple: These are exactly the permutations that do not contain the pattern 231. Since 1985, when the first systematic study was published by Simion and Schmidt [131], the area of permutation patterns has become a rapidly growing field of discrete mathematics, more specifically of combinatorics, as witnessed by monographs of Bóna [32] and Kitaev [105]. Many applications of permutation patterns have been discovered: their relation to stack and deque sorting, genome sequences in computational biology, statistical mechanics and in general their numerous connections to other combinatorial objects [105].

One of the most prominent examples of results concerning permutation patterns is the well-known Marcus–Tardos theorem (also known as the Stanley–Wilf conjecture). It states that the number of permutations that avoid a given pattern grows only single-exponentially. This is in stark contrast to the total number of permutations which grows super-exponentially. This theorem was proven by Marcus and Tardos [115] by proving a different conjecture by Füredi and Hajnal [84], which was shown to imply the Stanley–Wilf conjecture by Klazar [106].

While permutation patterns have been studied extensively from a combinatorial point of view, less is known about algorithmic aspects. The fundamental computational problem is the following: given an n-permutation T and a k-permutation P (the pattern), is P contained in T? We call this problem PERMUTATION PATTERN MATCHING, short PPM. PPM is known to NP-complete, as shown by Bose, Buss and Lubiw [34].

The most relevant algorithmic paper is the recent break-through result by Guillemot and Marx [92] showing that PPM is fpt with respect to the length of the pattern k. Their algorithm has a runtime of $2^{\mathcal{O}(k^2 \cdot logk)} \cdot n$. This fpt result is anteceeded by XP-algorithms with a runtime of $\mathcal{O}(n^{1+2k/3} \cdot \log n)$ [2] (Albert et al.) and $\mathcal{O}(n^{0.47k+o(k)})$ [1] (Ahal and Rabinovich).

Even though PPM is NP-complete in the general case, there are special cases of input instances for which the problem can be solved efficiently, i.e., in polynomial time. In the following, we list the cases for which it is known that PPM can be solved in polynomial time.

- In case the pattern is a *separable* permutation, i.e., a permutation avoiding both 3142 and 2413, PPM can be solved in $\mathcal{O}(k \cdot n^6)$, as shown by Bose, Buss and Lubiw [34]. This runtime has been improved to $\mathcal{O}(k \cdot n^4)$ by Ibarra [99]. For separable permutation patterns, also a polynomial-time, parallel algorithm has been designed by Saxena and Yugandhar [128].

- In case P is the identity $12 \ldots k$, PPM consists of looking for an increasing subsequence of length k in the text – this is a special case of the LONGEST INCREASING SUBSEQUENCE problem. This problem can be solved in $\mathcal{O}(n \log n)$-time for sequences in general, as proven by Schensted [129], and in $\mathcal{O}(n \log \log n)$-time for permutations, as shown by Chang and Wang [49] and Mäkinen [114].

- A $\mathcal{O}(k^2 n^6)$-time algorithm is presented by Guillemot and Vialette [93] for the case that both the text and the pattern are 321-avoiding.

- PPM can also be restricted by requiring that the pattern has to be matched to consecutive elements in the text. For this restriction, a $\mathcal{O}(n + k)$ algorithm has been found by Kubica et al. [110]. A similar result has been found independently by Kim et al. [104]. This work has been extended to the cases where some mismatches are tolerated, by Gawrychchowski and Uznanski [87]. Also an analogon of suffix trees has recently been developed for consecutive patterns by Crochemore et al. [58].

The related LONGEST COMMON PATTERN problem is to find a longest common pattern between two permutations T_1 and T_2, i.e., a pattern P of maximal length that can be matched both into T_1 and T_2. This problem is a generalization of PPM since determining whether the longest common pattern between T_1 and T_2 is T_1 is equivalent to PPM. Bouvel and Rossin [37] found a polynomial time algorithm for the LONGEST COMMON PATTERN problem for the case that one of the two permutations T_1 and T_2 is separable. A generalization of this problem, the so-called LONGEST COMMON $\mathcal{C}$-PATTERN problem was introduced by Bouvel, Rossin and Vialette [39]. This problem consists of finding the longest common pattern that belongs to a class of permutations $\mathcal{C}$. For the case that $\mathcal{C}$ is the class of all separable permutations and that the number of input permutations is fixed, the problem was shown to be polynomial-time solvable [39].

For a class of permutations X, the LONGEST X-SUBSEQUENCE (LXS) problem is to identify in a given permutation T its longest subsequence that is isomorphic to a permutation of X. Polynomial time algorithms for many classes X exist, but in general LXS is NP-complete, as shown by Albert et al. [3].

3.2 Structure in Preferences

Preferences appear throughout artificial intelligence in diverse applications. Consequently, preferences are the topic of many monographs [83, 126] and survey articles [53, 62, 91]. Our work on structure in preferences is mostly situated in social choice theory, more specifically in computational social choice. The aim of social choice is to develop a formal theory of joint decision making and voting. Computational social choice deals with the computational aspects of joint decision making. In these fields, preferences naturally arise since each participant (voter) has to specify their preferences in order to impact the decision.

While voting has been studied for centuries (notably Condorcet in the 18th century), modern social choice theory was established by the work of Arrow, in particular his impossibility theorem [7,8], published in 1950. Arrow's impossibility theorem states that every voting system that is based on cardinal preferences, i.e., preferences given as total orders, and yields a ranking of candidates cannot satisfy the following four properties:

- It has an unrestricted domain, i.e., every preference may occur as a vote.
- It satisfies the "independence of irrelevant alternatives" property, i.e., the relative ranking of two candidates is not influenced by a third candidate.
- It satisfies "Pareto efficiency", i.e., if every voter prefers candidate a over candidate b, then a has to be ranked above b.
- It satisfies "non-dictatorship", i.e., there is not a single voter (a dictator) that decides the outcome of any election.

This negative result was then extended to voting systems that only yield a single winner by Gibbard [89] and Satterthwaite [127]. The corresponding theorem, the so-called Gibbard–Satterthwaite theorem, shows that every voting system choosing a single winner cannot satisfies the following three properties:

- It is possible for every candidate to win.
- The voting system is immune to tactical voting, i.e., the voters never have an incentive to misreport their true preferences.
- There is no dictator.

These two fundamental results establish boundaries of any voting system and thus force us to find different voting systems for different scenarios; a perfect voting system cannot exist.

While social choice is a well-established field of research with an extensive amount of literature (as witnessed by a wealth of monographs [4,8–10,33,85,118]), computational social choice is a rather new field [42]. One of the founding research publications was *Voting schemes for which it can be difficult to tell who won the election* by Bartholdi, Tovey and Trick in 1989 [21]. In this work, the authors analyze the computational complexity of a voting rule where it is NP-hard to determine a winner. Another foundational work is *The computational difficulty of manipulating an election* [20] from the same authors, also published 1989. Here, the authors study the computational complexity of manipulating an election, that is, strategic voting. This paper was the first to propose computational complexity as a shield against dishonest voting behavior and thus can be seen as a positive answer to the problems raised by the Gibbard-Satterthwaite theorem: if manipulation is possible, it might be computationally infeasible to compute a strategy that manipulates the outcome of an election.

Computational social choice has identified voting as a very general and useful method of collective decision-making and preference aggregation. Voting applications have been found in many settings ranging from politics to artificial intelligence (in particular multi-agent systems) and other topics in computer science such as rank aggregation on the web [65], planning [69], database systems [133] or recommender systems [88, 123]. We would now like to highlight publications in computational social choice that are related to the topic of structure in preferences.

Single-peaked preferences. The single-peaked restriction [30] was introduced by Black in 1948. Escoffier, Lang, and Ötztürk [71] presented a linear-time algorithm for deciding whether a given profile is single-peaked, improving upon previous work by Bartholdi and Trick [23]. Ballester and Haeringer [16] combinatorially characterize single-peaked elections by the means of forbidden configurations; this characterization will play an important role in Chapter 7.

Computationally hard problems often become easier for single-peaked preferences, for example winner determination problems, proportional representation, manipulation and control [29, 41, 75, 76]. Conitzer [52] considers single-peaked preferences also in the context of preference elicitation. Single-peaked preferences have also been extended to a multi-dimensional setting by Barberà et al. [17]. Practical and algorithmic aspects of this extension have been studied by Sui, Francois-Nienaber and Boutilier [135].

Other domain restrictions. Other well-known examples of domain restrictions are single-caved preferences (Inada [100]), single-crossing preferences (Mirrlees [119]), value-restricted preferences (Sen [130]), and group-separable preferences (Inada [100, 101]). Many of these domains enjoy desirable social choice-theoretic properties, such as transitivity of the majority relation and existence of a strategyproof social choice rule, as shown by Barberá and Moreno [18]. All these domain restrictions can be characterized by forbidden configurations: the characterization of the single-peaked, single-caved and group-separable domain was shown by Ballester and Haeringer [16], the characterization of the single-crossing domain by Bredereck, Chen and Woeginger [45]. A characterization by forbidden configurations immediately yields a detection algorithm requiring only polynomial time (cf. Proposition 7.1). Even faster algorithms exist for single-crossing electorates, designed by Elkind, Faliszewski and Slinko [67] and Bredereck, Chen and Woeginger [45].

While the single-peaked domain has received most attention, also single-crossing elections have proven to be an algorithmically advantageous structure. For example, single-crossing preferences have been studied in the context of proportional representation by Skowron, Faliszewski and Elkind [132].

Another domain restriction is single-peakedness on trees [60, 136, 143], where preferences, in contrast to single-peaked preferences, are not ordered on a linear axis but on a tree. Also, top monotonicity has been recently introduced which is a relaxation of several domain restrictions such as the single-crossing and single-peaked domain [13, 18, 19].

Nearly structured preferences. Domain restrictions are often too restrictive for real-world data. Thus, notions of distance have been considered recently. Faliszewski, Hemaspaandra, and Hemaspaandra [75] analyzed the complexity of bribery, control, and manipulation in nearly single-peaked elections. Single-peaked width has been been studied by Cornaz, Galand, and Spanjaard in the context of Kemeny winner determination [57] and proportional representation [56]. Elkind, Faliszewski, and Slinko [67] define the decloning measure which describes the number of adjacent candidates (adjacent in every vote) that are merged into one candidate in order to obtain single-peakedness. Bredereck, Chen, and Woeginger [44] study two distance measures (maverick and candidate deletion) for several domain restrictions.

Incomplete preferences. If only incomplete preferences are available, it might be impossible to determine the winner of an election. In such a scenario it makes sense to distinguish possible and necessary winners. Computational questions of determining possible and necessary winners were first studied by Konczak and Lang [71]. Further work by Baumeister and Rothe [26], Baumeister et al. [25], Lang et al. [112], Xia and Conitzer [142], Pini et al. [124] and Betzler and Dorn [27] has shown that possible and necessary winner computation given incomplete preferences is NP-hard for many voting systems that allow for polynomial-time winner computation in case of complete information. Betzler et al. [28] employ the framework of parameterized complexity to find fast parameterized algorithms.

Another question in the context of incomplete preferences is preference elicitation which has been studied by Conitzer and Sandholm [54] and Walsh [138–140].

Part I

Permutation Patterns

CHAPTER 4

Permutation Pattern Matching for Generalized Patterns

This chapter is based on the publication *The computational landscape of permutation patterns* [47], a joint work with Marie-Louise Bruner.

In recent years, several types of generalized permutation patterns have received increased interest, such as vincular [14], bivincular [35], mesh [40], boxed mesh [12] and consecutive patterns [66] (all of which are introduced in Section 4.1). Every type of permutation pattern naturally defines a corresponding computational problem. Let $\mathcal{C}$ denote any type of permutation pattern, i.e., let $\mathcal{C} \in \{$classical, vincular, bivincular, mesh, boxed mesh, consecutive$\}$.

$\mathcal{C}$ PERMUTATION PATTERN MATCHING ($\mathcal{C}$ PPM)	
Instance:	A permutation T (the text) and a $\mathcal{C}$ pattern P.
Question:	Does the $\mathcal{C}$ pattern P occur in T?

In this chapter we study the classical, vincular, bivincular, mesh, boxed mesh and consecutive pattern matching problem. Often we abbreviate CLASSICAL PERMUTATION PATTERN MATCHING with PPM and the other problems with $\mathcal{C}$ PPM, where $\mathcal{C}$ is the corresponding pattern type.

This chapter draws a map of the computational landscape of permutation patterns and thus aims at paving the way for a detailed computational analysis. Its contents are the following:

- We survey different types of permutation patterns (Section 4.1), focusing on classical, vincular, bivincular, mesh, boxed mesh and consecutive patterns. The hierarchy of these patterns with the most general one at the top is displayed in Figure 4.1.

- We study the computational complexity of each corresponding permutation pattern matching problem. It is known that CLASSICAL PERMUTATION PATTERN MATCHING is NP-complete [34] and consequently VINCULAR, BIVINCULAR and MESH PPM are NP-hard as well. We strengthen this result and also show that pattern matching with boxed mesh and consecutive patterns can be performed in polynomial time (Section 4.2).

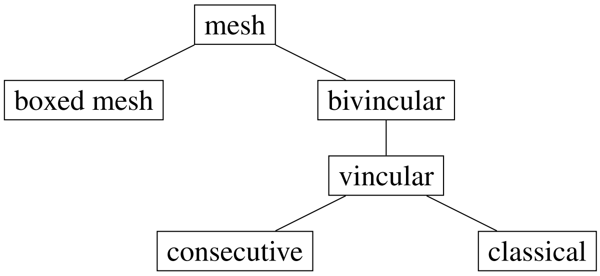


Figure 4.1: Hierarchy of pattern types

- We offer a more fine-grained complexity analysis by employing the framework of parameterized complexity. For most NP-complete problems we provide a more detailed complexity classification by showing W[1]-completeness with respect to the parameter length of P (Section 4.3). Both the classical as well as the parameterized complexity results are summarized in Table 4.1 (page 25) and Table 4.2 (page 26).

4.1 Types of Patterns

In this section we give an overview of several different types of permutation patterns that have been introduced in the last years and that will be of interest in this chapter. These are classical, vincular, bivincular, mesh, boxed mesh and consecutive patterns. A schematic representation of their hierarchy can be found in Figure 4.1. For details, we refer the reader to the Chapters 1 and 5-7 in Kitaev's monograph *Patterns in Permutations and Words* [105]. Before we introduce different types of patterns, we precisely define matchings in the context of permutation patterns.

Definition 4.1. *Let $\mathcal{C} \in \{$classical, vincular, bivincular, mesh, boxed mesh, consecutive$\}$. A* matching *of a $\mathcal{C}$ pattern P of length k into a permutation T of length n is an increasing mapping $M : [k] \to [n]$ such that the sequence $M(P(1)), M(P(2)), \ldots, M(P(k))$ is an occurrence of the $\mathcal{C}$ pattern P in T.*

4.1.1 Classical Patterns

Classical permutation patterns, or simply permutation patterns, have implicitly been studied in different contexts for more than a hundred years. The first mentioning of a (classical) permutation pattern is attributed to Knuth and an exercise of his *Fundamental algorithms* [109] in 1968. It was however only in 1985 that Simion and Schmidt performed the first systematic study of patterns in permutations in [131]. To put the definition of pattern containment with classical patterns in relation to pattern containment of other types of patterns, we repeat the definition of pattern containment (Definition 2.1) in a slightly different formulation.

Definition 4.2. *Let P be a k-permutation and $T = T(1) \ldots T(n)$ an n-permutation. An* occurrence of the classical permutation pattern *P is a subsequence $T(i_1)\,T(i_2)\,\ldots\,T(i_k)$ of T that is*

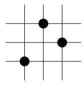
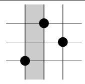
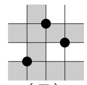
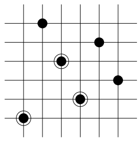
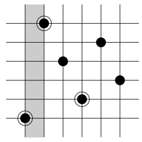
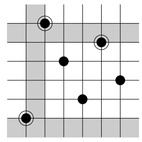
	Classical	Vincular	Bivincular
Pattern	$P = 132 =$ 	$P = \underline{13}2 =$  cols$(P) = 1$	$P = \begin{smallmatrix} \ulcorner 1\overline{23} \\ \underline{13}2 \end{smallmatrix} =$  cols$(P) = 1$ rows$(P) = 2$
Text			
Classical complexity	NP-complete [34]	NP-complete Corollary 4.3	NP-complete Corollary 4.3
Parameterized complexity	FPT [92]	W[1]-complete Theorem 4.6	W[1]-complete Theorem 4.7

Table 4.1: Examples of classical, vincular and bivincular permutation patterns

order-isomorphic to P, i.e., a subsequence in which the letters appear in the same relative order as in P. If such a subsequence exists, one says that T contains *P or that there is a* matching *of P into T. If there is no such function one says that T* avoids *the (classical) pattern P.*

Example 4.1. The classical pattern $P = 132$ is contained several times in the text $T = 164253$ as for instance shown by the subsequence 142. A matching M is given by $M(1) = 1$, $M(3) = 4$ and $M(2) = 2$. The pattern $P = 1234$ is however not contained in T since no increasing subsequence of length four can be found in T. $\dashv$

Graphically, a permutation π on $[n]$ can be represented with the help of a $[0, n+1] \times [0, n+1]$-grid in which elements marked by black circles are placed at the position (i, j) whenever $\pi(i) = j$. This representation thus corresponds to the function graph of π when viewing permutations as bijective maps. Representing permutations with the help of grids allows for a simple interpretation of classical pattern containment respectively avoidance in permutations. Indeed, the pattern P is contained in the permutation T if and only if the grid corresponding to P can be obtained from the one corresponding to T by deleting some columns and rows. For the example given above, see the left column in Table 4.1, in which the elements involved in the matching have been marked by circled elements.

It is easy to see that P can be matched into T if and only if P^c can be matched into T^c, if and only if P^r can be matched into T^r and if and only if P^{-1} can be matched into T^{-1}.

4.1.2 Vincular Patterns

Let $T(i_1)T(i_2)\dots T(i_k)$ be an occurrence of the classical pattern P in the text T. Then there are no requirements on the elements in T lying in between $T(i_j)$ and $T(i_{j+1})$. It is however natural to ask for occurrences of patterns in which certain elements are forced to be adjacent in the text, i.e., $T(i_{j+1}) = T(i_j + 1)$. Vincular patterns are a generalization of classical patterns capturing these requirements on adjacency in the text. They were introduced under the name of *generalized*

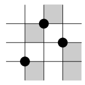
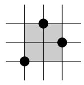
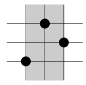
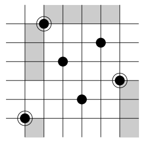
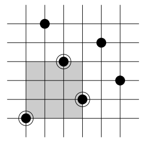
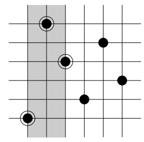
	Mesh	Boxed mesh	Consecutive
Pattern	$P = (\pi, R) =$  $\mathsf{cells}(P) = 5$	$P = \boxed{132} =$ 	$P = \underline{132} =$ 
Text			
Classical complexity	NP-complete Corollary 4.3	in P; Theorem 4.4	in P; Theorem 4.5
Parameterized Complexity	W[1]-complete Theorem 4.8	trivially FPT	trivially FPT

Table 4.2: Examples of mesh, boxed mesh and consecutive permutation patterns

patterns in 2000 by Babson and Steingrímsson in [14], where it was shown that essentially all Mahonian permutation statistics in the literature can be written as linear combinations of vincular patterns. For a survey of this topic, see [134].

Here we use the name of *vincular patterns* as it was introduced by Kitaev in [105]. We also use the notation introduced there, since it is consistent with the notation for classical patterns.

Definition 4.3. *A* vincular pattern P *is a permutation in which certain consecutive entries may be underlined. An occurrence of P in a permutation T is then an occurrence of the corresponding classical pattern for which underlined elements are matched to adjacent elements. To be more formal: An occurrence of P in T corresponds to a subsequence $T(i_1)T(i_2)\dots T(i_k)$ of T that is order-isomorphic to P and for which $T(i_{j+1}) = T(i_j + 1)$ whenever P contains $\underline{P(j)P(j+1)}$. Furthermore, if P starts with $\llcorner P(1)$ an occurrence of P in T must start with the first entry in T, i.e., $T(i_1) = T(1)$. Similarly, if P ends with $P(k)\lrcorner$ it must hold that $T(i_k) = T(n)$.*

When representing permutations by grids, adjacency of positions clearly corresponds to adjacency of columns. In order to represent the underlined elements in vincular patterns in the corresponding grids, one shades the columns which may not contain any elements in a matching. For an example, see the middle column of Table 4.1. Matching the pattern $\underline{13}2$ into the permutation T, means that no elements may lie in the columns between $M(1)$ and $M(3)$ in T.

In order to specify how many adjacency restrictions are made in the vincular pattern P, we define $\mathsf{cols}(P)$ to be the number of shaded columns in the grid corresponding to P.

Note that the operations *complement* and *reverse* may be performed on vincular patterns, leading to some (other) vincular pattern. Similarly as for classical patterns it then holds that P can be matched into T if and only if P^c can be matched into T^c and if and only if P^r can be matched into T^r. The inverse of a vincular pattern is however not clearly defined. This leads to a larger class of patterns which is introduced below.

4.1.3 Bivincular Patterns

Bivincular patterns generalize classical patterns even further than vincular patterns. Indeed, in bivincular patterns, not only positions but also values of elements involved in a matching may be forced to be adjacent. When Bousquet-Mélou, Claesson, Dukes and Kitaev introduced bivincular patterns in 2010 [35], the main motivation was to find a minimal superset of vincular patterns that is closed under the inverse operation. As mentioned in Section 4.1.2, the inverse of a vincular pattern is not well-defined - it is a bivincular, but not a vincular pattern.

Definition 4.4. *A* bivincular pattern P *is a permutation written in two-line notation, where some elements in the top row may be overlined and the bottom row is a vincular pattern as defined in Definition 4.3. An occurrence* $T(i_1)T(i_2)\dots T(i_k)$ *of* P *in a permutation* T *is an occurrence of the corresponding vincular pattern where additionally the following holds:* $T(i_{j+1}) = T(i_j)+1$ *whenever the top row of* P *contains* $\overline{j(j+1)}$*. Furthermore, if the top row starts with* $\overline{\vert 1}$*, an occurrence of* P *in* T *must start with the smallest entry in* T*, i.e.,* $T(i_1) = 1$*. Similarly, if the top row ends with* $\overline{k\vert}$*, it must hold that* $T(i_k) = n$*.*

This definition gets a lot less cumbersome when representing permutations with the help of grids: As remarked earlier, underlined elements in the bottom row are translated into forbidden columns in which no elements may occur in a matching. Similarly, overlined elements in the top row are translated into forbidden rows. For an example, see the right column in Table 4.1.

Again, in order to specify how many adjacency restrictions are made in the bivincular pattern P, we define - in addition to $\mathsf{cols}(P)$ - $\mathsf{rows}(P)$ to be the number of shaded rows in the grid corresponding to P.

4.1.4 Mesh Patterns

A further generalization of bivincular patterns was given by Brändén and Claesson who introduced mesh patterns in [40] in 2011. Mesh patterns allow further restrictions on the relative positions of the entries in an occurrence of a pattern. Several permutation statistics can be formulated as the number of occurences of certain mesh patterns [40].

Definition 4.5. *A* mesh pattern *is a pair* $P = (\pi, R)$ *where* π *is a permutation of length* k *and* $R \subset [0,k] \times [0,k]$ *is a relation. An occurrence of* P *in a permutation* T *is an occurrence of the classical pattern* π *fulfilling additional restrictions defined by* R*. That is to say there is a subsequence* $T(i_1)T(i_2)\dots T(i_k)$ *of* T *that is order-isomorphic to* π *and the following property holds:*

$$(x,y) \in R \Longrightarrow \nexists i \in [n] : i_x < i < i_{x+1} \wedge T\left(i_{\pi^{-1}(y)}\right) < T(i) < T\left(i_{\pi^{-1}(y+1)}\right).$$

This definition is again a lot easier to capture when representing permutations as grids. Indeed, the relation R can be translated very easily into the graphical representation of $P = (\pi, R)$, by shading the unit square with bottom left corner (x, y) for every $(x, y) \in R$. An occurrence of P in a permutation T is then a classical occurrence of π in T such that no elements of T lie in the shaded regions of the grid.

Again, in order to specify how many adjacency restrictions are made in the mesh pattern P, we define $\mathsf{cells}(P)$ to be the number of shaded cells in the corresponding grid. Thus $\mathsf{cells}(\pi, R) := |R|$. For an example consider the mesh pattern $P = (\pi, R)$ with $\pi = 132$ and $R = \{(1,0), (1,2), (2,3), (3,0), (3,1)\}$ which is displayed in the left column of Table 4.2.

4.1.5 Boxed Mesh Patterns

A special case of mesh patterns, so called boxed mesh patterns, was very recently introduced by Avgustinovich, Kitaev and Valyuzhenich in [12].

Definition 4.6. *A* boxed mesh *pattern, or simply boxed pattern, is a mesh pattern $P = (\pi, R)$ where π is a permutation of length k and $R = [1, k-1] \times [1, k-1]$. P is then denoted by $\boxed{\pi}$.*

In the grid representing a boxed pattern all but the boundary squares are shaded. For an example, see the middle column of Table 4.2.

It is straightforward to see that the set of boxed patterns is closed under taking complements, reverses and inverses and that these operations are compatible with pattern containment. Interestingly, it was shown [12] that the statement "π can be matched into T if and only if $\boxed{\pi}$ can be matched into T" is only true if π is one of the following permutations: 1, 12, 21, 132, 213, 231, 312.

4.1.6 Consecutive Patterns

Consecutive patterns are a special case of vincular patterns, namely those where *all* entries are underlined. In an occurrence of a consecutive pattern it is thus necessary that all entries are adjacent. Finding an occurrence of a consecutive pattern therefore consists in finding a contiguous subsequence of T that is order-isomorphic to P. For an example, see the right column of Table 4.2.

Several well-known enumeration problems for permutations can be formulated in terms of forbidden consecutive patterns; Elizalde and Noy [66] provide examples. Chapter 5 in [105] is devoted to and gives an overview of different methods employed in the literature for the study of consecutive patterns.

4.2 The Possibility of Polynomial-Time Algorithms

4.2.1 NP-completeness

At the 1992 SIAM Discrete Mathematics meeting Herbert Wilf asked whether it is possible to solve the permutation pattern matching problem in polynomial time. The answer is no unless P=NP, as shown by the NP-completeness result of Bose, Buss and Lubiw [34]. This result immediately yields NP-hardness for all generalizations of classical permutation pattern matching. In this section we are going to show that NP-hardness holds for these problems even in a more restricted case: with all runs having length at most two.

Definition 4.7. *A* run *in a permutation is a maximal monotone contiguous subsequence. Let* $\mathsf{lrun}(\pi)$ *denote the length of the longest run in the permutation* π.

Note that for any permutation π with length at least two it holds that $\mathsf{lrun}(\pi) \geq 2$.

Theorem 4.1. *Every* MESH PERMUTATION PATTERN MATCHING *instance* (P,T) *with* $P = (\pi, R)$ *can be transformed into an instance* (P',T') *with* $P' = (\pi', R)$ *and the following properties:* (P',T') *is a yes-instance if and only if* (P,T) *is yes-instance,* $|\pi'| = 2|\pi|$, $|T'| = 2|T|$ *and* $\mathsf{lrun}(\pi') = \mathsf{lrun}(T') = 2$. *This transformation can be done in polynomial time.*

Proof. Let $\pi = p_1 \ldots p_k$ and $T = t_1 \ldots t_n$. We define

$$\pi' = (k+1)\, p_1\, (k+2)\, p_2\, (k+3) \ldots (2k)\, p_k$$
$$T' = (n+1)\, t_1\, (n+2)\, t_2\, (n+3) \ldots (2n)\, t_n.$$

Clearly, $|\pi'| = 2|\pi|$, $|T'| = 2|T|$ and $\mathsf{lrun}(\pi') = \mathsf{lrun}(T') = 2$. We are now going to show that there is a matching from P into T if and only if there is a matching from P' into T'. Assume that M is a matching from P into T, i.e., a function from $[k]$ to $[n]$. We extend this function to a function M' from $[2k+1]$ to $[2n+1]$ in the following way:

$$M'(i) = \begin{cases} M(i), & \text{if } i \in [k], \\ T(j), \text{ where } M(i-k) = T(j+1) & \text{if } i > k. \end{cases}$$

In other words, M' maps $(i+k)$ to the element in T left of $M(i)$. For example if $M(p_3) = t_5$ then $p_3 \in \pi'$ is matched to $t_5 \in T'$ and $(k+3) \in \pi'$ is matched to $n+5 \in T'$ (which is the element in T lying directly to the left of t_5). Observe that the function M' is a matching from P' into T'.

Now let us assume that M' is a matching from P' into T'. If we restrict the domain of M' to $[k]$ then we obtain a matching from P into T. □

Theorem 4.2. PERMUTATION PATTERN MATCHING *is* NP*-complete even on permutations* P *and* T *with* $\mathsf{lrun}(P') = \mathsf{lrun}(T') = 2$.

Proof. We apply the transformation in Theorem 4.1 to show NP-hardness. NP-membership holds for this restricted class of input instances as well. □

Corollary 4.3. VINCULAR, BIVINCULAR *and* MESH PPM *are* NP*-complete even if* $\mathsf{lrun}(P') = \mathsf{lrun}(T') = 2$.

Proof. NP-hardness follows from Theorem 4.1 as well as from Theorem 4.2. NP-membership holds since checking whether the additional restrictions imposed by the vincular, bivincular or mesh pattern are fulfilled can clearly be done in polynomial time. □

4.2.2 Polynomial time algorithms

We have seen that polynomial time algorithms are unlikely to exist for PPM and its generalizations. However, this is not the case for the special cases of boxed mesh and consecutive pattern matching.

Theorem 4.4. BOXED MESH PERMUTATION PATTERN MATCHING *can be solved in* $\mathcal{O}(n^3)$ *time.*

Proof. Let P be a boxed pattern of length k and T a permutation of length n. For every pair (i, j) where $i \in [n]$ and $i + k \leq j \leq n$ check whether there is a matching M of the *boxed* pattern P into T where the smallest element in P is matched to i and the largest one to j, i.e., $M(1) = i$ and $M(k) = j$.

Checking whether such a matching exists can be done in the following way: From the permutation T, construct the permutation $\tilde{T}$ by deleting all elements that are smaller than i and larger than j. Clearly, the matching that we are looking for must be contained in $\tilde{T}$, it could otherwise not be an occurrence of a boxed pattern. Moreover, it has to consist of k *consecutive* elements in $\tilde{T}$. Since the positions of the smallest and the largest element are fixed, the positions for all other elements of P are equally determined. Thus there is only one subsequence of T that could possibly be a matching of P into T with $M(1) = i$ and $M(k) = j$. Deleting the elements that are too small or too large and checking whether this subsequence actually corresponds to an occurrence of P in T, i.e., whether it is order-isomorphic to P, can be done in at most n steps. Note that this subsequence might consist of less than k elements in which case it clearly does not correspond to an occurrence.

In total, there are $(n - k + 1) \cdot (n - k + 2)/2 = \mathcal{O}(n^2)$ pairs (i, j) that have to be checked which leads to the runtime bound $\mathcal{O}(n^3)$. □

Theorem 4.5. CONSECUTIVE PERMUTATION PATTERN MATCHING *can be solved in* $\mathcal{O}((n - k) \cdot k)$ *time.*

Proof. Let P be a consecutive pattern of length k and T a permutation of length n. For every $i \in [n - k + 1]$ check whether there is a matching of P into T where the first element of P is mapped to i. Since we are looking for an occurrence of a consecutive pattern, the only possible subsequence of T then consists of the element i and the following $(k - 1)$ elements of T. Whether this sequence is order-isomorphic to P can be checked in k steps which leads to the runtime bound $\mathcal{O}((n - k) \cdot k)$. □

As has recently been shown by Kubica et.al. in [110], this simple result can be improved by an algorithm with runtime $\mathcal{O}(n + k)$.

4.3 The Impact of the Pattern Length

PPM can be solved in $\mathcal{O}(n^k)$ time by exhaustive search, where k is the length of P. This trivial upper bound has been improved first by Albert et al. to $\mathcal{O}(n^{1+2k/3} \cdot \log n)$ [2] and then to $\mathcal{O}(n^{0.47k+o(k)})$ by Ahal and Rabinovich [1]. In a recent breakthrough result, Guillemot and

Marx have shown that PPM can be solved by an FPT algorithm [92]. Its runtime is $2^{\mathcal{O}(k^2 \cdot \log k)} \cdot n$. In this section we are going to show that such a result is likely not to be achievable for VINCULAR, BIVINCULAR and MESH PPM. This is done by showing W[1]-hardness with respect to the parameter k. First, we show that MESH PPM and therefore all other problems studied in this chapter are contained in W[1].

All results in this section are summarized in Figure 4.4 on page 37.

Theorem 4.6. MESH PERMUTATION PATTERN MATCHING *is contained in* W[1].

Proof. For showing membership we encode MESH PPM as a model checking problem of an existential first order formula. W[1]-membership is then a consequence of the fact that the following problem is W[1]-complete [79].

EXISTENTIAL FIRST-ORDER MODEL CHECKING	
Instance:	A structure $\mathcal{A}$ and an existential first-order formula φ
Parameter:	$\lvert\varphi\rvert$
Question:	Is $\mathcal{A}$ a model for φ?

Let $((P, R), T)$ be a MESH PPM instance. We compute a structure $\mathcal{A} = (A, <, \prec_T, E)$, where the domain set $A = \{1, \ldots, n\}$ represents indices in the text. The binary relation $\prec_T$ is defined by $x \prec_T y$ holds if and only if $T(x) < T(y)$. E is a quaternary relation where $E(w, x, y, z)$ is true if and only if there are no elements in T that are left of w, right of x, larger than y and smaller than z. Intuitively, w, x, y and z describe a *forbidden* rectangle in the permutation grid of T which may not contain any elements of T. $T_<$, E and $<$ can be computed in polynomial time. The formula φ we want to check is

$$\varphi = \exists x_1 \ldots \exists x_k \quad x_1 < x_2 \;\wedge\; x_2 < x_3 \;\wedge\; \ldots \;\wedge\; x_{k-1} < x_k \;\wedge$$
$$\underbrace{\bigwedge_{\substack{P(i)<P(j)\\ \text{for } i,j\in[k]}} x_i \prec_T x_j \wedge}_{\varphi_1} \underbrace{\bigwedge_{\substack{P(i)>P(j)\\ \text{for } i,j\in[k]}} \neg(x_i \prec_T x_j) \wedge}_{\varphi_2} \underbrace{\bigwedge_{\substack{i,j\in[k] \text{ and}\\ R(i,j) \text{ is true.}}} E(x_i, x_{i+1}, x_j, x_{j+1})}_{\varphi_3}.$$

Observe that the length of φ is in $\mathcal{O}(k^2)$. The two sub-formulas φ_1 and φ_2 are exactly then true when a subsequence $T(x_1)T(x_2)\ldots T(x_k)$ of T can be found such that $T(x_i) < T(x_j)$ if and only if $P(i) < P(j)$. Thus $\varphi_1 \wedge \varphi_2$ is true if and only if there is a matching of the classical pattern P into T. The sub-formula φ_3 encodes the relation R and is true if and only if no elements lie in the forbidden regions of T, as can be seen by recalling Definition 4.5. Thus φ is true if and only if $((P, R), T)$ is a yes-instance of MESH PPM. □

We now want to prove W[1]-hardness for vincular, bivincular and mesh pattern matching. For this purpose, we introduce here SEGREGATED PERMUTATION PATTERN MATCHING, a generalization of PPM. All subsequent hardness theorems use reductions from this problem.

SEGREGATED PERMUTATION PATTERN MATCHING (SPPM)		
	Instance:	A permutation T (the text) of length n, a permutation P (the pattern) of length $k \leq n$ and two positive integers $p \in [k]$, $t \in [n]$.
	Parameter:	k
	Question:	Is there a matching M of P into T such that $M(i) \leq t$ if and only if $i \leq p$?

Example 4.2. Consider the pattern $P = 132$ and the text $T = 53142$. As shown by the matching $M(2) = 3$, $M(1) = 1$ and $M(3) = 4$, the instance $(P, T, 2, 3)$ is a yes-instance of the SPPM problem. However, $(P, T, 2, 4)$ is a NO-instance, since no matching of P into T can be found where $M(3) > 4$. $\dashv$

Theorem 4.7. SEGREGATED PERMUTATION PATTERN MATCHING *is* W[1]*-hard with respect to the parameter* k.

Proof. We show W[1]-hardness by giving an fpt-reduction from the W[1]-complete CLIQUE problem [63] to SPPM:

CLIQUE		
	Instance:	A graph $G = (V, E)$ and a positive integer k.
	Parameter:	k
	Question:	Is there a subset of vertices $S \subseteq V$ of size k such that S forms a clique, i.e., the induced subgraph $G[S]$ is complete?

The reduction has three parts. First, we will show that we are able to reduce a CLIQUE instance to a pair (P', T'), where P' and T' are two permutations on multisets, i.e., permutations in which elements may occur more than once. Applying Definition 4.2 to permutations on multisets means that in a matching repeated elements in the pattern have to be mapped to repeated elements in the text. In addition to repeated elements, P' and T' contain so-called *guard elements*. Their function is explained below. Second, we will show how to get rid of repetitions. The method used in this step has already been used in the NP-completeness proof of PPM provided by Bose, Buss and Lubiw in [34]. Third, we implement the guards by using the segregation property and have thus reduced CLIQUE to SPPM.

Let (G, k) be a CLIQUE instance, where $V = \{v_1, v_2, \ldots, v_l\}$ is the set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ the set of edges. Both the pattern and the text consist of a single substring coding vertices ($\dot{P}$ resp. $\dot{T}$) and substrings coding edges ($\bar{P}_i$ resp. $\bar{T}_i$ for the i-th substring). These substrings are listed one after the other, with *guard elements* placed in between them. These guard elements have the function of separating substrings in a matching: guard elements will have to be mapped to guard elements and substrings embraced by two consecutive guard-elements will also have to be mapped to substrings embraced by two consecutive guard-elements. For the moment, we will simply write brackets to indicate where guard elements are placed. The meaning of these brackets is then the following: a block of elements enclosed by a $\langle$ to the left and a $\rangle$ to the right has to be matched into another block of elements between two such brackets. How the guard-elements are implemented as elements of a permutation is explained at the end of the proof after Claim 2.

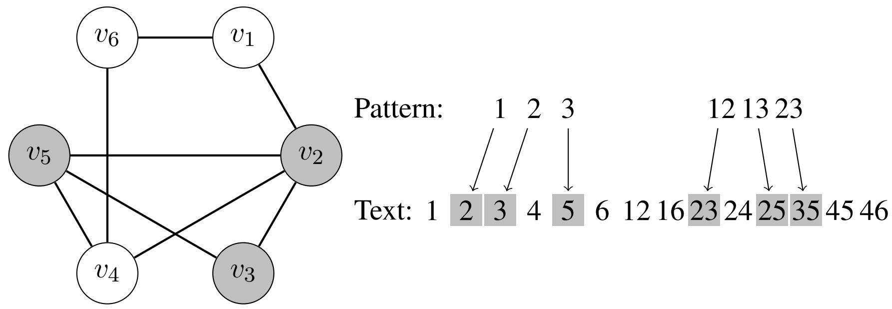


Figure 4.2: An example for the reduction of an INDEPENDENT SET instance to a PPM instance

We define the pattern to be

$$\begin{aligned} P' &:= \langle \dot{P} \rangle \langle \bar{P}_1 \rangle \langle \bar{P}_2 \rangle \langle \ldots \rangle \langle \bar{P}_{k(k-1)/2} \rangle \\ &= \langle 123 \ldots k \rangle \langle 12 \rangle \langle 13 \rangle \langle \ldots \rangle \langle 1k \rangle \langle 23 \rangle \langle \ldots \rangle \langle 2k \rangle \langle \ldots \rangle \langle (k-1)k \rangle . \end{aligned}$$

$\dot{P}$ corresponds to a list of (indices of) k vertices. The $\bar{P}_i$'s represent all possible edges between the k vertices (in lexicographic order).

For the text

$$T' := \langle \dot{T} \rangle \langle \bar{T}_1 \rangle \langle \bar{T}_2 \rangle \langle \ldots \rangle \langle \bar{T}_m \rangle$$

we proceed similarly. $\dot{T}$ is a list of the (indices of the) l vertices of G. The $\bar{T}_i$'s represent all edges in G (again in lexicographic order). Let us give an example:

Example 4.3. Let $l = 6$ and $k = 3$. Then the pattern permutation is given by

$$P' = \langle 123 \rangle \langle 12 \rangle \langle 13 \rangle \langle 23 \rangle .$$

Consider for instance the graph G with six vertices $v_1, \ldots, v_6$ and edge-set

$$\{\{1,2\}, \{1,6\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,5\}, \{4,5\}, \{4,6\}\} .$$

represented in Figure 4.2 (we write $\{i, j\}$ instead of $\{v_i, v_j\}$).

Then the text permutation is given by:

$$T' = \langle 123456 \rangle \langle 12 \rangle \langle 16 \rangle \langle 23 \rangle \langle 24 \rangle \langle 25 \rangle \langle 35 \rangle \langle 45 \rangle \langle 46 \rangle .$$

$\dashv$

Claim 1. *A clique of size k can be found in G if and only if there is a simultaneous matching of $\dot{P}$ into $\dot{T}$ and of every $\bar{P}_i$ into some $\bar{T}_j$.*

Example 4.4 (continuation)**.** In our example $\{v_2, v_3, v_5\}$ is a clique of size three. Indeed, the pattern P' can be matched into T' as can be seen by matching the elements $1, 2$ and 3 onto $2, 3$ and 5 respectively. See again Figure 4.2 where the involved vertices respectively elements of the text permutation have been marked in gray. $\dashv$

Proof of Claim 1. A matching of $\dot{P}$ into $\dot{T}$ corresponds to a selection of k vertices amongst the l vertices of G. If it is possible to additionally match every one of the $\bar{P}$'s into a $\bar{T}$ this means that all possible edges between the selected vertices appear in G. This is because T' only contains pairs of indices that correspond to edges appearing in the graph. The selected k vertices thus form a clique in G. Conversely, if for every possible matching of $\dot{P}$ into $\dot{T}$ defined by a monotone function $M : [k] \to [l]$ some $\bar{P}_i = xy$ cannot be matched into T', this means that $\{M(x), M(y)\}$ does not appear as an edge in G. Thus, for every selection of k vertices there will always be at least one pair of vertices that are not connected by an edge and therefore there is no clique of size k in G. $\square$

In order to get rid of repeated elements, we identify every variable with a real interval: 1 corresponds to the interval $[1, 1.9]$, 2 to $[2, 2.9]$ and so on until finally k corresponds to $[k, k+0.9]$ (resp. l to $[l, l + 0.9]$). In $\dot{P}$ and $\dot{T}$ we shall therefore replace every element j by the pair of elements $(j + 0.9, j)$ (in this order). The occurrences of j in the $\bar{P}_i$'s (resp. $\bar{T}_i$'s) shall then successively be replaced by real numbers in the interval $[j, j+0.9]$. For every j, these values are chosen one after the other (from left to right), always picking a real number that is larger than all the previously chosen ones in the interval $[j, j + 0.9]$.

Observe the following: The obtained sequence is not a permutation in the classical sense since it consists of real numbers. However, by replacing the smallest number by 1, the second smallest by 2 and so on, we do obtain an ordinary permutation. This defines P and T (except for the guard elements).

Example 4.5 (continuation)**.** Getting rid of repetitions in the pattern of the above example could for instance be done in the following way:

$$P = \langle 1.9\ 1\ 2.9\ 2\ 3.9\ 3\rangle\langle 1.1\ 2.1\rangle\langle 1.2\ 3.1\rangle\langle 2.2\ 3.2\rangle$$

This permutation of real numbers is order-isomorphic to the following ordinary permutation:

$$P = \langle 4\,1\,8\,5\,12\,9\rangle\langle 2\,6\rangle\langle 3\,10\rangle\langle 7\,11\rangle.$$

$\dashv$

Claim 2. *P can be matched into T if and only if P' can be matched into T'.*

Proof of Claim 2. Suppose that P' can be matched into T'. When matching P into T, we have to make sure that elements in P that were copies of some repeated element in P' may still be mapped to elements in T that were copies themselves in T'. Indeed this is possible since we have chosen the real numbers replacing repeated elements in increasing order. If i in P' was matched to j in T', then the pair $(i + 0.9, i)$ in P may be matched to the pair $(j + 0.9, j)$ in T and the increasing sequence of elements in the interval $[i, i + 0.9]$ may be matched into the increasing sequence of elements in the interval $[j, j + 0.9]$.

Now suppose that P can be matched into T. In order to prove that this implies that P' can be matched into T', we merely need to show that elements in P that were copies of some repeated element in P' have to be mapped to elements in T that were copies themselves in T'. Then returning to repeated elements clearly preserves the matching. Firstly, it is clear that a pair of consecutive elements $i + 0.9$ and i in $\dot{P}$ has to be matched to some pair of consecutive elements $j + 0.9$ and j in $\dot{T}$, since j is the only element smaller than $j + 0.9$ and appearing to its right. Thus intervals are matched to intervals. Secondly, an element x in P for which it holds that $i < x < i + 0.9$ must be matched to an element y in T for which it holds that $j < y < j + 0.9$. Thus copies of an element are still matched to copies of some other element.

Finally, replacing real numbers by integers does not change the permutations in any relevant way. □

It remains to implement the guards in order to ensure that substrings are matched to corresponding substrings. Let $P_{\max}$ and $T_{\max}$ denote the largest integer that is contained in P respectively T at this point. We now replace all guards with integers larger than $P_{\max}$ respectively $T_{\max}$ and will choose the *segregating elements* p and t such that guards and "original" pattern/text elements are separated. We insert the guard elements in the designated positions (previously marked by $\langle$ and $\rangle$) in the following order: $P_{\max} + 2$ (instead of the first $\langle$), $P_{\max} + 1$ (instead of the first $\rangle$), $P_{\max} + 4$ (instead of the second $\langle$), $P_{\max} + 3$ (instead of the second $\rangle$), $\ldots$, $P_{\max} + 2i$ (instead of the i-th $\langle$), $P_{\max} + 2i - 1$ (instead of the i-th $\rangle$), $\ldots$, and so on until we reach the last guard-position. The guard elements are inserted in this specific order to ensure that two neighboring guard elements $\langle$ and $\rangle$ in P have to be mapped to two neighboring guard elements $\langle$ and $\rangle$ in T. We proceed analogously in T. To ensure that guards in P are matched to guards in T and pattern elements of P are matched to text elements in T, we set p to $P_{\max}$ and t to $T_{\max}$.

This finally yields that (G, k) is a yes-instance of CLIQUE if and only if (P, T) is a yes-instance of SPPM. It can easily be verified that this reduction can be done in fpt-time. □

As can easily be seen, the reduction performed in the proof of Theorem 4.7 can be done in polynomial time. Thus this proof immediately yields NP-hardness for SPPM.

Now, that we have obtained this result, we are able to show $\mathsf{W[1]}$-hardness for PPM with vincular, bivincular and mesh patterns. As before, the parameter is the length of the pattern.

Theorem 4.6. VINCULAR PERMUTATION PATTERN MATCHING *is* $\mathsf{W[1]}$*-complete with respect to* k*. This holds even when restricting the problem to instances* (P, T) *with* $\mathsf{cols}(P) = 1$.

Proof. We reduce from SEGREGATED PPM. Let (P, T, p, t) be an SPPM instance. The VINCULAR PPM instance (P', T') constructed from (P, T) will have have an additional element in P' and an additional element in T'. The new element in P, denoted by p', is $p + 0.5$, i.e., p' is larger than p but smaller than $p + 1$. Analogously, $t' = t + 0.5$ is the new element in T. We define $P' = \llcorner p' P$ and $T' = t'\ T$. In order to obtain a permutation P on $[k + 1]$ and T on $[n + 1]$, we simply need to relabel the respective elements order-isomorphically. In every matching of P' into T' the element p' has to be mapped to t'. Consequently, all elements larger than p' in P' have to be mapped to elements larger than t' in T' and all elements smaller than p' have to be mapped to elements smaller than t'. This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance if and only if (P', T') is a VINCULAR PPM yes-instance. This reduction is done in linear time which proves $\mathsf{W[1]}$-hardness of VINCULAR PPM. Membership follows from Theorem 4.6. □

Theorem 4.7. BIVINCULAR PERMUTATION PATTERN MATCHING *is* W[1]*-complete with respect to k. This holds even when restricting the problem to instances (P, T) with* rows$(P) = 1$.

Proof. As in the previous proof we reduce from SEGREGATED PPM. Let (P, T, p, t) be an SPPM instance. Identically to the previous proof, we define $p' = p + 0.5$ and $t' = t + 0.5$. The BIVINCULAR PPM instance consists of a permutation P' with elements in $[k+1] \cup \{p'\}$ and T', a permutation on $[n+1] \cup \{p'\}$. We define

$$P' = \begin{array}{ccccccc} 1 & 2 & 3 & \dots & p' & \dots & \overline{(k{+}1)]} \\ p' & (k+1) & P(1) & & \dots & & P(k) \end{array}$$

and $T' = t'(n+1)T$. In order to obtain permutations on $[k+2]$ respectively $[n+2]$ we again relabel the elements order-isomorphically.

In any matching of P' into T' the element $(k+1)$ has to be mapped to $(n+1)$ and therefore p' has to be mapped to t'. Thus all elements larger than p' in P' have to be mapped to elements larger than t' in T' and all elements smaller than p' have to be mapped to elements smaller than t'. This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance if and only if (P', T') is a BIVINCULAR PPM yes-instance. Since this reduction can again be done in linear time, BIVINCULAR PPM is W[1]-hard. Membership follows again from Theorem 4.6. □

Theorem 4.8. MESH PERMUTATION PATTERN MATCHING *is* W[1]*-complete with respect to k. This holds even if* cells$(P) = 1$.

Proof. Let (P, T, p, t) be a SEGREGATED PPM instance. As before, we define $p' = p + 0.5$ and $t' = t + 0.5$. The MESH PPM instance consists of a permutation P' with elements in $[k] \cup \{p'\}$ and T', a permutation on $[n+1] \cup \{p'\}$. Again, permutations on $[k+1]$ respectively $[n+2]$ can be obtained by relabelling the elements order-isomorphically. We define $P' = p'\ P$ and $T' = t'\ (n+1)\ T$. Furthermore, let $R = \{(0, (k+1))\}$. This means that for every matching M of P' into T' the following must hold: to the left of $M(p')$ in T', there are no elements larger than $M(k)$. However, it surely holds that $M(k) \leq (n+1)$. Consequently, p' has to be mapped to t'. This implies that (P, T, p, t) is a SEGREGATED PPM yes-instance if and only if (P', T') is a MESH PPM yes-instance. Since this reduction can again be done in linear time, MESH PPM is W[1]-hard. Membership follows from Theorem 4.6. □

These hardness results show that we cannot hope for a fixed-parameter tractable algorithm for VINCULAR/BIVINCULAR/MESH PERMUTATION PATTERN MATCHING.

4.4 Summary

In this chapter, we have strengthened the previously known NP-hardness result for PPM and proved NP-completeness for its generalizations. We have also found polynomial time algorithms for boxed mesh and consecutive PPM. See Figure 4.3 for an overview of the classical complexity of PPM with generalized patterns. Furthermore, we have performed a parameterized complexity analysis for the parameter k, the pattern length. We showed that for vincular, bivincular and

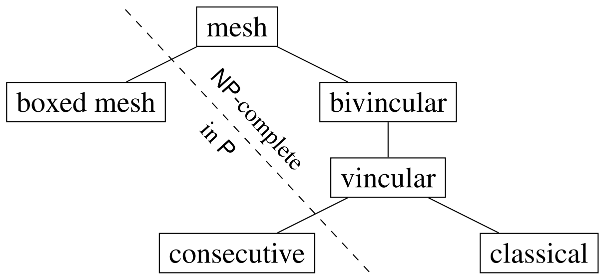


Figure 4.3: Classical complexity of permutation pattern matching with different pattern types

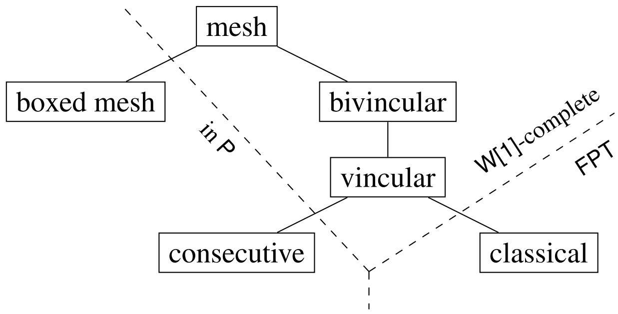


Figure 4.4: The influence of the pattern length on the computational hardness: parameterized complexity of permutation pattern matching

mesh PPM a fixed-parameter tractable algorithm is unlikely to exist. This is in contrast to the case of classical PPM, which is fpt with respect to k [92]. Refer to Figure 4.4 for an overview of these parameterized results.

CHAPTER 5

Fast Permutation Pattern Matching

This chapter is based on the publication *A fast algorithm for permutation pattern matching based on alternating runs* [46], a joint work with Marie-Louise Bruner. Here, our focus is on classical permutation patterns. The corresponding pattern matching problem is defined as follows:

PERMUTATION PATTERN MATCHING (PPM)	
Instance:	A permutation T (the text) of length n and a permutation P (the pattern) of length $k \leq n$.
Question:	Is there a matching of P into T?

Bose, Buss and Lubiw [34] showed that PPM is in general NP-complete. The trivial brute-force algorithm checking every subsequence of length k of T has a runtime of $\mathcal{O}(2^n \cdot n)$. So far, no algorithm has been discovered that improves the exponential runtime to c^n for some constant $c < 2$. Yet, improving exponential time algorithms is a major topic in algorithmics, as witnessed by the monograph of Fomin and Kratsch [81].

In this chapter we tackle the problem of solving PPM faster than $\mathcal{O}(2^n \cdot n)$ for arbitrary P and T. Our algorithm has a runtime of $\mathcal{O}(1.79^n \cdot n \cdot k)$. We achieve this by exploiting the decomposition of permutations into alternating runs. As an example, the permutation $\pi = 53142$ has three alternating runs: 531 (down), 4 (up) and 2 (down). We denote this number of ups and downs in a permutation π by $\mathsf{run}(\pi)$. Alternating runs are a fundamental permutation statistic and were studied already in the late 19th century by André [5]. Despite the importance of alternating runs within the study of permutations, the connection to PPM has so far not been explored. For a detailed summary of results, the reader is referred to Section 5.3.

5.1 The Alternating Run Algorithm

We start with an outline of the alternating run algorithm. Its description consists of two parts. In Part 1 we introduce so-called *matching functions*. These functions map runs in P to sequences of adjacent runs in T. The intention behind matching functions is to restrict the search space to certain subsequences of length k, namely to those where all elements in a run in P are mapped to elements in the corresponding sequences of runs in T. In Part 2 a dynamic programming algorithm is described. It checks for every matching function whether it is possible to find a compatible matching. This is done by finding a small set of representative elements to which the element 1 can be mapped to, then – for a given choice for 1 – finding representative values for 2, and so on.

Theorem 5.1. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}(1.79^{\mathsf{run}(T)} \cdot n \cdot k)$. *Therefore,* PPM *parameterized by* $\mathsf{run}(T)$ *is in* FPT.

Since $\mathsf{run}(T) < n$, we obtain as an immediate consequence:

Corollary 5.2. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}(1.79^n \cdot n \cdot k)$.

Before we start with the description of the alternating run algorithm, we introduce two functions which play an important role.

Definition 5.1. *Let* $i \in [k]$. *The* run predecessor $\mathsf{pre}(i)$ *denotes the largest element smaller than* i *that is contained in the same run as* i *in* P *(if such an element exists). Moreover, the* run index *function* ri *is defined as follows:* $\mathsf{ri}(i) = j$ *if* i *is contained in the* j*-th run in* P.

Note that both functions concern only the pattern P.

5.1.1 Matching Functions

We introduce the concept of matching functions. These are functions from the interval $[\mathsf{run}(P)]$ to sequences of adjacent runs in T. For a given matching function F the search space in T is restricted to matchings where an element j contained in the i-th run in P is matched to an element in $F(i)$. As we will see later on in Lemma 5.3, this restriction of the search space does not influence whether a matching can be found or not: if a matching exists, a corresponding matching function can be found. In addition, Lemma 5.11 will show that it is possible to iterate over all matching functions in fpt time. Thus, our algorithm verifies for all matching functions whether a compatible matching exists.

Let us now give a formal definition of matching functions.

Definition 5.2. *A* matching function F *maps an element of* $[\mathsf{run}(P)]$ *to a subsequence of* T. *It has to satisfy the following properties for all* $i \in [\mathsf{run}(P)]$.

(P1) $F(i)$ *is a contiguous subsequence of* T.

(P2) If the i*-th run in* P *is a run up (down),* $F(i)$ *starts with an element following a valley (peak) or the first element in* T *and ends with a valley (peak) or the last element in* T.

(P3) $F(1)$ *starts with the first and* $F(\mathsf{run}(P))$ *ends with the last element in* T.

(P4) $F(i)$ *and* $F(i+1)$ *have one run in common:* $F(i+1)$ *starts with the leftmost element in the last run in* $F(i)$.

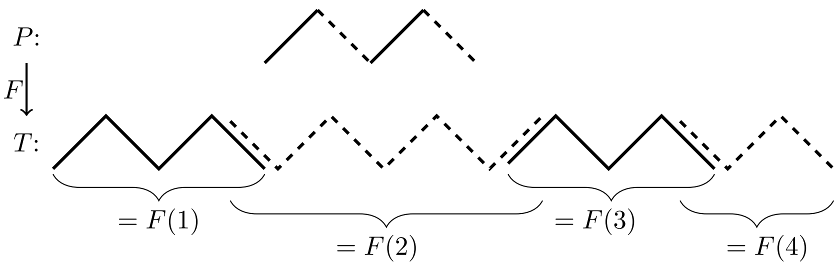


Figure 5.1: A sketch of a matching function and its M- and W-shaped subsequences

Property (P2) implies that every run up is matched into an M-shaped sequence of runs of the form up–down–up–...–up–down (if the run up is the first or the last run in P the sequence might start or end differently) and every run down is matched into a W-shaped sequence of runs of the form down–up–down–...–down–up (again, if the run down is the first or the last run in P, the sequence might start or end differently). These M- and W-shaped sequences are sketched in Figure 5.1.

Property (P4) implies that two adjacent runs in P are mapped to sequences of runs that overlap with exactly one run, as is also sketched in Figure 5.1. This overlap is necessary since elements in different runs in P may be matched to elements in the same run in T. More precisely, valleys and peaks in P might be matched to the same run in T as their successors (see the following example).

Example 5.1. Throughout this chapter we will use the text permutation

$$T_{ex} = 1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$$

and the pattern permutation $P_{ex} = 2\,3\,1\,4$ as a running example. In Figure 5.2, P_{ex} (left-hand side) and T_{ex} (right-hand side) are depicted together with a matching function F. A matching compatible with F is given by $4\,6\,2\,9$. We can see that the elements 6 and 2 lie in the same run in T_{ex} even though 3 (a peak) and 1 (its successor) lie in different runs in P_{ex}. $\dashv$

Note that there are no matching functions if $\mathsf{run}(P) > \mathsf{run}(T)$. This corresponds to the fact that in such a case no matching from P into T exists either. The properties (P1)-(P4) guarantee that the number of functions we have to consider is less than $(\sqrt{2})^{\mathsf{run}(T)}$, as will be proven in Section 5.1.4, Lemma 5.11. This allows us to iterate over all matching functions in fpt time.

Let us formalize what we mean by compatible matchings.

Definition 5.3. *A matching M is* compatible *with a matching function F if $M(\kappa) \in F(\mathsf{ri}(\kappa))$ for every $\kappa \in [k]$, i.e., M matches each element contained in the i-th run in P to an element in $F(i)$.*

Lemma 5.3. *For every matching M of P into T there exists a matching function F such that M is compatible with F.*

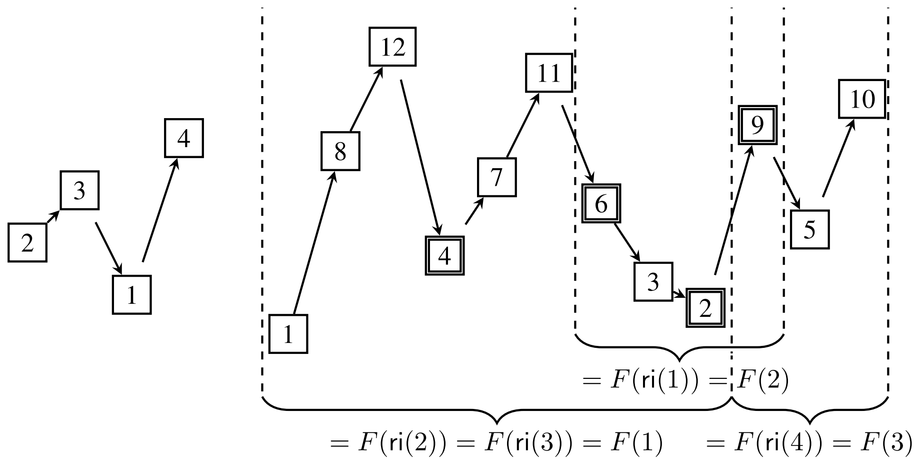


Figure 5.2: P_{ex} and T_{ex} together with a matching function F and the compatible matching witnessed by the subsequence 4 6 2 9

The proof of this lemma can be found in Section 5.1.3. We continue with the observation that, when searching for a compatible matching by looking for the possible values that $M(1), M(2)$ and so on can take, we do not have to remember *all* the previous choices we made. Let us have a look at an example first:

Example 5.2. In Figure 5.2, assume that we already have a partial matching: $M(1) = 2$ and $M(2) = 4$. We now have to decide where to map 3. There are two constraints that have to be satisfied: First, $M(3) > M(2)$. Second, $M(3)$ has to be to the right of $M(2)$, since $2 \prec_P 3$. Since our choices for $M(3)$ are limited to $F(3)$, we do not have to check whether $M(3)$ is left of $M(1)$ but only whether $M(3) > M(2)$. Later, when deciding where to map 4, we will only have to verify that $M(4) > M(3)$.

In more generality, we observe that given a matching function and a partial matching M defined on $[\kappa - 1]$, deciding where to map κ only requires the knowledge of $M(\kappa - 1)$ and of $M(\kappa')$, where κ' is the previous element in the same run as κ. $\dashv$

Let us now make this observation more precise:

Lemma 5.4. *Let F be a matching function. A function $M{:}[k] \to [n]$ is a matching of P into T compatible with F if and only if for every $\kappa \in [k]$:*

1. *$M(\kappa) \in F(\mathsf{ri}(\kappa))$,*

2. *$M(\kappa) > M(\kappa - 1)$ and*

3. *if $\mathsf{pre}(\kappa)$ exists, then $\mathsf{pre}(\kappa) \prec_P \kappa$ if and only if $M(\mathsf{pre}(\kappa)) \prec_T M(\kappa)$, i.e., if κ is contained in a run up (down), then $M(\kappa)$ is right (left) of $M(\mathsf{pre}(\kappa))$.*

As we will see soon, this lemma is essential for our algorithm. Its proof can be found in Section 5.1.3.

5.1.2 Algorithm Description

Before we start explaining the actual fpt algorithm, let us consider a simple algorithm based on alternating runs. This simple algorithm (Algorithm 1) does not have fpt runtime but has the same basic structure as the fpt algorithm. In particular, this simple algorithm will already demonstrate the importance of Lemma 5.4.

Algorithm 1: A Simple Alternating Run Algorithm

$X_0^F \leftarrow \{(0,\dots,0)\}$ `// The tuple` $(0,\dots,0)$ `has` $\mathsf{run}(P)$ `elements.`
foreach *matching function* F **do**
  for $\kappa \leftarrow 1 \dots k$ **do** `//` κ `is the element to be matched.`
    $X_\kappa^F \leftarrow \emptyset$
    foreach $\vec{x} \in X_{\kappa-1}^F$ **do**
      $R \leftarrow \{\nu \in [n] : \nu \in F(\mathsf{ri}(\kappa)) \wedge \nu > x_{\mathsf{ri}(\kappa-1)} \wedge (\mathsf{pre}(\kappa) \prec_P \kappa \leftrightarrow x_{\mathsf{ri}(\mathsf{pre}(\kappa))} \prec_T \nu)\}$ `// Conditions according to Lemma 5.4`
      foreach $\nu \in R$ **do**
        $X_\kappa^F \leftarrow X_\kappa^F \cup \{(x_1,\dots,x_{\mathsf{ri}(\kappa)-1},\nu,x_{\mathsf{ri}(\kappa)+1},\dots,x_{\mathsf{run}(P)})\}$
  if $X_k^F \neq \emptyset$ **then**
    return *"P can be matched into T."*
return *"P cannot be matched into T."*

From Lemma 5.3 we know that when checking whether T contains P as a pattern, it is sufficient to test for all matching functions whether there exists a *compatible* matching. Let us fix a matching function F. We first find suitable elements to which 1 can be mapped, then suitable elements for 2, and so on. Observe that we can use Lemma 5.4 to verify what suitable elements are. In addition, Lemma 5.4 tells us that when finding suitable elements for $\kappa \in [k]$, we only require the values of $M(\kappa-1)$ and $M(\mathsf{pre}(\kappa))$. This means in particular that we do not have to store all values of a possible partial matching $(M(1),\dots,M(\kappa))$ but only the values of M for the largest element $\leq \kappa$ in each run in P. For example, when trying to match $P = 2\,3\,5\,7\,4\,1\,6$ into some text and looking for the possible values for $\kappa = 4$, we only have to consider possibilities for $M(3)$ and $M(\mathsf{pre}(4)) = M(1)$.

In this simple algorithm, we want to keep track of all possible partial matchings $(M(1),\dots,M(\kappa))$ for every $\kappa \in [k]$. Since such partial matchings can be described by storing a single value per run in P, every one of them can be stored as a tuple $\vec{x}$ of length $\mathsf{run}(P)$. The first element of $\vec{x}$ contains a possible choice for the largest element $\leq \kappa$ in the first run of P, the second element of $\vec{x}$ contains a possible choice for the largest element $\leq \kappa$ in the second run of P, etc. We formalize this notion of "tuples encoding partial matchings" as (κ, F)-matchings:

Definition 5.4. *Let κ be an integer in $[k]$. A tuple $\vec{x} = (x_1, x_2, \dots, x_{\mathsf{run}(P)})$ with $x_i \in [0,n]$ for all $i \in [\mathsf{run}(P)]$ is called a (κ, F)-matching of P into T if the following holds: There exists* *a function $M : [\kappa] \to [n]$ that is a matching of $P|_{[\kappa]}$ into T that is compatible with F and for which it additionally holds that for every $x_i \neq 0$, $M(\max\{\kappa' \leq \kappa : \mathsf{ri}(\kappa') = i\}) = x_i$, i.e., M maps the largest element $\leq \kappa$ in the i-th run of P to the i-th element of $\vec{x}$.*

The following lemma states that X_κ^F – as constructed by Algorithm 1 – indeed contains only tuples that are (κ, F)-matchings:

Lemma 5.5. *Let X_κ^F be the set of tuples as constructed by Algorithm 1. Then every $\vec{x} \in X_\kappa^F$ is a (κ, F)-matching.*

The proof can be found in Section 5.1.3. As an immediate consequence of this lemma, we know that if $X_k^F \neq \emptyset$ then there exists a matching from P into T that is compatible with F. Observe that X_k^F is always empty if a previous X_κ^F was empty. If for every F the set $X_k^F = \emptyset$, we know from Lemma 5.3 that P cannot be matched into T.

Example 5.3. For our running example (P_{ex}, T_{ex}) and $\kappa = 1$ the data structure is given as follows: $X_1^F = \{(0,6,0),(0,3,0),(0,2,0),(0,9,0)\}$. Given the choice $M(1) = 3$, we obtain 6 $(2, F)$-matchings, namely: $(8,3,0)$, $(12,3,0)$, $(4,3,0)$, $(7,3,0)$, $(11,3,0)$ and $(6,3,0)$. In total X_2^F contains 19 elements. $\dashv$

As seen in this small example, the set R and consequently the set X_κ^F can get very large. In particular, it is not possible to bound the size of X_κ^F by a function depending only on $\mathsf{run}(T)$ and not on n – which is necessary for obtaining our fpt result. Thus, we have to further refine our algorithm.

We proceed by explaining how this simple algorithm can be improved in order to obtain an fpt algorithm based on alternating runs (Algorithm 2). This is the main algorithm described in this chapter. In the following description we fix F to be the current matching function under consideration. There are two modifications that have to be made in order to obtain fpt runtime. First, we have to restrict the set R to fewer, *representative* choices. Second, we have to change the data structure of X_κ^F from a set to an array of fixed size. In the array X_κ^F, every (κ, F)-matching has a predetermined position. Observe that if there are two (κ, F)-matchings $\vec{x}$, $\vec{y}$ where $\vec{x}$ leading to a matching implies that $\vec{y}$ leads to a matching as well, the algorithm only has to remember $\vec{y}$. The position of a (κ, F)-matching will thus be assigned in such a way that one of two (κ, F)-matching sharing the same position is preferable in the above sense. We will now explain both modifications in detail.

Concerning the first modification, i.e., restricting the set R, we introduce the procedure $\texttt{Rep}(\vec{x}, \kappa, F)$. This procedure returns a set of representative elements to which κ can be mapped. These choices have to be compatible with previously chosen elements $(x_1, x_2, \ldots, x_{\mathsf{run}(P)})$ and the matching function F.

An element $\nu \in [n]$ is contained in $\texttt{Rep}(\vec{x}, \kappa, F)$ if the following conditions are met:

(C1) [Line 1] It has to hold that $\nu \in F(\mathsf{ri}(\kappa))$ (cf. Condition 1 in Lemma 5.4).

(C2) [Line 2] It has to hold that $\nu > x_{\mathsf{ri}(\kappa-1)}$ (cf. Condition 2 in Lemma 5.4).

Algorithm 2: The Alternating Run Algorithm

1 $X_0^F \leftarrow [(0,\ldots,0)]$ `// (0,...,0) has` $\mathsf{run}(P)$ `elements.`
2 **foreach** *matching function* F **do**
3   **for** $\kappa \leftarrow 1 \ldots k$ **do** `//` κ `is the element to be matched.`
4         $X_\kappa^F \leftarrow [\epsilon,\ldots,\epsilon]$ `//` X_κ^F `is a fixed-size array.`
5   **foreach** $\vec{x} \in X_{\kappa-1}^F$ *with* $\vec{x} \neq \epsilon$ **do**
6             $R \leftarrow \texttt{Rep}(\vec{x},\kappa,F)$
7   **foreach** $\nu \in R$ **do**
8                 $i \leftarrow \texttt{Index}(x_1,\ldots,x_{\mathsf{ri}(\kappa)-1},\nu,x_{\mathsf{ri}(\kappa)+1},\ldots,x_{\mathsf{run}(P)})$
9                 $\vec{y} \leftarrow X_\kappa^F(i)$
10   **if** $\vec{y} = \epsilon$ **or** $y_{\mathsf{ri}(\kappa)} > \nu$ **then**
11                     $X_\kappa^F(i) \leftarrow (x_1,\ldots,x_{\mathsf{ri}(\kappa)-1},\nu,x_{\mathsf{ri}(\kappa)+1},\ldots,x_{\mathsf{run}(P)})$
12   **if** $X_k^F \neq [\epsilon,\ldots,\epsilon]$ **then** `// Is` X_k^F `non-empty?`
13   **return** *"Matching found:* $\texttt{GetMatching}(X_1^F,\ldots,X_k^F)$*"*
14 **return** *"P cannot be matched into T."*

(C3) [Line 3] It is always preferable to choose elements that are as small as possible. To be more precise: If we consider the subsequence of T containing all elements in the set R, we merely need to consider the valleys of this subsequence. The function $\texttt{Valleys}(T|_R)$ returns exactly these valleys.

(C4) [Lines 6 and 13] It has to hold that if κ is contained in a run up (down), then ν has to be right (left) of $x_{\mathsf{ri}(\kappa)}$, i.e., the element to which the run predecessor of κ is mapped (cf. Condition 3 in Lemma 5.4).

(C5) [Lines 8 and 15] If κ is the largest element in its run, the optimal choice is the smallest possible element.

(C6) [Lines 10 and 17] If κ is not the largest element in its run, the choice of ν must not prevent finding elements for the next elements in its run. Thus, if κ is contained in a run up (down), then there has to be a larger element to its right (left) that is contained in $F(\mathsf{ri}(\kappa))$.

Since this smaller set R is a subset of the set R in the simple algorithm (Algorithm 1), we immediately obtain the following corollary of Lemma 5.5:

Corollary 5.6. *Let X_κ^F be the set of tuples as constructed by Algorithm 2. Then every $\vec{x} \in X_\kappa^F$ is a (κ, F)-matching.*

Example 5.4. Let us explain how the elements in $\texttt{Rep}((4,2,0),3,F)$ are determined in our running example. The elements fulfilling Condition (C1) are: 1, 8, 12, 4, 7, 6, 3 and 2 (listed in the order they appear in T). Among these, the elements larger than $x_{\mathsf{ri}(2)} = x_1 = 4$ are: 8, 12, 7, 11, 6 (cf. (C2)). If we consider this subsequence, its valleys are: 8, 7, and 6 (these are the elements fulfilling Condition (C3)). The element 3 is contained in a run up in T, thus the element it is mapped to has to lie to the right of $x_{\mathsf{ri}(\mathsf{pre}(3))} = x_{\mathsf{ri}(2)} = 4$. The elements also

Procedure Rep($\vec{x}, \kappa, F$)

input : a (κ, F)-matching $\vec{x} = (x_1, x_2, \ldots, x_{\mathsf{run}(P)})$, $\kappa \in [k]$, a matching function F
output: R, the set of representative elements for $M(\kappa)$

$R \leftarrow F(\mathsf{ri}(\kappa))$
$R \leftarrow R \cap [x_{\mathsf{ri}(\kappa-1)} + 1, n]$
$R \leftarrow \texttt{Valleys}(T|_R)$
if κ *is in a run up in* P **then**
  if $x_{\mathsf{ri}(\kappa) \neq 0}$ **then**
    $R \leftarrow \left\{\nu \in R : x_{\mathsf{ri}(\kappa)} \prec_T \nu\right\}$
  if κ *is the largest element in its run* **then**
    $R \leftarrow \{\min R\}$
  else
    $R \leftarrow \{\nu \in R : \exists \nu'$ with $\nu' \in F(\mathsf{ri}(\kappa)) \wedge \nu' > \nu \wedge \nu \prec_T \nu'\}$
else
  if $x_{\mathsf{ri}(\kappa) \neq 0}$ **then**
    $R \leftarrow \left\{\nu \in R : \nu \prec_T x_{\mathsf{ri}(\kappa)}\right\}$
  if κ *is the largest element in its run* **then**
    $R \leftarrow \{\min R\}$
  else
    $R \leftarrow \{\nu \in R : \exists \nu'$ with $\nu' \in F(\mathsf{ri}(\kappa)) \wedge \nu' > \nu \wedge \nu' \prec_T \nu\}$
return R

fulfilling (C4) thus are 7 and 6. Since 3 is the largest element in its run in P, we only need to store the smallest possibility which is 6 (cf. (C5)). Condition (C6) does not apply here. If there were another, larger element in the same run as 3 in P, we would have to choose the element 7, since there are no larger elements in $F(\mathsf{ri}(3))$ to the right of 6. ⊣

If any matching of P into T can be found that is compatible with F, it is also possible to find a matching that only involves representative elements. This statement is formalized and proven in Section 5.1.3 (Definition 5.6 and Lemma 5.7). For the time being, let us convey the intuition behind this:

Example 5.5. In Figure 5.2, $4\,6\,3\,10$ is a matching of P_{ex} into T_{ex} where the elements 3 and 10 are not representative: $3 \notin \texttt{Rep}((0,0,0), 1, F)$ and $10 \notin \texttt{Rep}((6,3,0), 4, F)$. This can be seen since 3 is not a valley in T and 10 is not a valley in the subsequence consisting of elements larger than 6. However, this matching can be represented by the matching $4\,6\,2\,9$ that only involves representative elements (3 is represented by 2; 10 by 9) and that is compatible with the same matching function F. ⊣

This concludes our description of representative elements, our first modification of the simple alternating run algorithm. We proceed by explaining the data structure X^F_κ, which is changed from a set to an array of fixed size. In this array, every (κ, F)-matching $\vec{x}$ has a predetermined position which depends on the notion of *vales*.

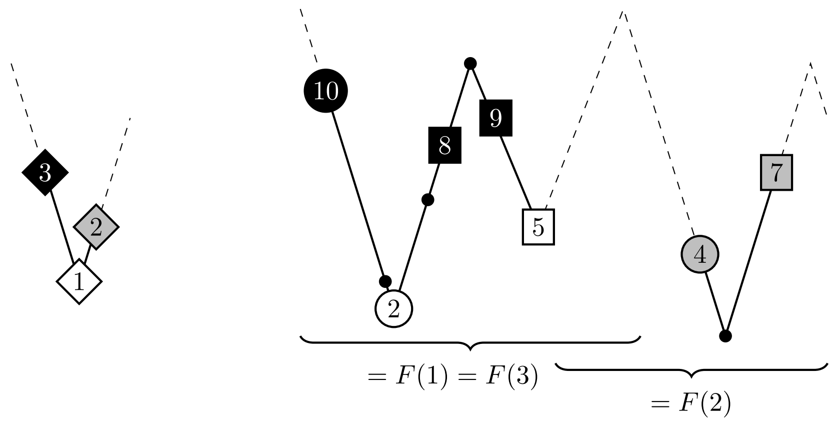


Figure 5.3: Schematic representation of the permutations occurring in Example 5.6: to the left is the pattern P, to the right is the text T.

Definition 5.5. *A subsequence of a permutation π consisting of a consecutive run down and run up (formed like a V) is called a* vale*. If π starts with a run up, this run is also considered as a vale and analogously if π ends with a run down. Let* $\mathsf{vale}(\pi)$ *denote the number of vales in π. Finally, we define the* vale index *function* $\mathsf{vi}(x)$*: given a matching function F and $x \in F(i)$, let* $\mathsf{vi}(x) = j$ *if x is contained in the j-th vale in $F(i)$. For notational convenience,* $\mathsf{vi}(0) = 1$.

The main idea is the following: Two (κ, F)-matchings $\vec{x}$ and $\vec{y}$ in X_κ^F with $v(x_i) = v(y_i)$ for all $i \in [\mathsf{run}(P)]$ are comparable in the sense that one of these is less likely to lead to a matching. More precisely, the (κ, F)-matching containing the larger element at the $\mathsf{ri}(\kappa)$-th position (this is also the largest element of the entire tuple) leads to a matching only if the other one leads to a matching as well. Thus, the former (κ, F)-matching can be discarded and only the latter (κ, F)-matching has to be stored. The following example illustrates this notion of comparability:

Example 5.6. Consider the two permutations P and T schematically represented in Figure 5.3. We are searching for representative elements for $\kappa = 3$ which lie in a run down in P. Which elements κ may be matched to depends on the choices for its run predecessor $\mathsf{pre}(3) = 1$ and for $\kappa - 1 = 2$. For the element 1, two representative elements are 2 (circle) and 5 (square), the valleys in $F(1)$ in T. They lead to one representative element for 2 each: if 2 has been chosen then 4 is a representative element (circle) and if 5 has been chosen then 7 (square) is one. At this point, we have the following two $(2, F)$-matchings: $\vec{x} = (\ldots, 0, 2, 4, 0, \ldots)$ and $\vec{y} = (\ldots, 0, 5, 7, 0, \ldots)$. On the one hand, $\vec{x}$ seems to be preferable since it involves smaller elements than $\vec{y}$ and this leaves more possibilities for the following elements. On the other hand, $\vec{y}$ seems to be preferable since it involves 5 in $F(1)$, which is further to the right than 2. This is advantageous since $F(1)$ corresponds to a run down and this means that larger elements in the same run will have to be chosen to the left. All together we cannot say which of $\vec{x}$ and $\vec{y}$ is preferable and thus have to store both of them.

When we now turn to the element 3 in P, there are three representative elements: if we have chosen $\vec{x}$ the only possible choice is the element 10; if we have chosen $\vec{y}$ there are two possible choices namely 8 and 9. We thus obtain three $(3, F)$ matchings: $\vec{x}' = (\ldots, 0, 10, 4, 0, \ldots)$, $\vec{y}' = (\ldots, 0, 8, 7, 0, \ldots)$ and $\vec{y}'' = (\ldots, 0, 9, 7, 0, \ldots)$. We can now observe that we do not have to keep track of all three possibilities. Indeed, the two $(3, F)$-matchings $\vec{x}'$ and $\vec{y}'$ have coinciding vales and $\vec{x}'$ can be discarded in favor of $\vec{y}'$ since $\vec{x}'$ will only lead to a matching of P into T if $\vec{y}'$ does. This is due to the fact that $x'_{\mathsf{ri}(3)} = 10 > 8 = y'_{\mathsf{ri}(3)}$ and can be seen as follows:

Let i be an element in the same run as 3 in P that is larger than 3 (which means that it lies to the left of 3). All elements to the left of and larger than 10 in $F(i)$ are clearly also to the left of and larger than 8. Thus, if there exists an element $\nu \in \texttt{Rep}(\vec{x}', i, F)$ there also exists a smaller element in $\texttt{Rep}(\vec{y}', i, F)$. This means that from the point of view of the run containing 3, $\vec{y}'$ is to be preferred over $\vec{x}'$. Now let $i > 3$ be an element in the same run in P as 2 (which means that it lies to the right of 2). Representative elements for i have to both lie to the right of the element chosen for 2 (4 or 7) and be larger than the element chosen for 3 (10 or 8). Since 4 and 7 lie in the same vale in T there are no larger elements in between them. This implies that elements that are to the right of 4 in $F(2)$ and larger than 10 are automatically to the right of 7 and larger than 8. From the point of view of the run containing 2, $\vec{y}'$ it also to be preferred over $\vec{x}'$. The same argument also holds for any other element i in P that is larger than 3.

To put this example a nutshell: if we have two (κ, F)-matchings $\vec{x}$ and $\vec{y}$ with coinciding vales and $y_{\mathsf{ri}(\kappa)} \leq x_{\mathsf{ri}(\kappa)}$ we only need to store $\vec{y}$. For a formal proof of this statement, we refer the reader to Lemma 5.9 in Section 5.1.3. $\dashv$

If we store only one (κ, F)-matching out of those with identical vales, the question arises how many vales there are in $F(i)$, $i \in [\mathsf{run}(P)]$. The answer is that at most $\lfloor \mathsf{run}(F(i))/2 \rfloor + 1$ exist: all vales but the two outermost consist of two runs and the two outermost may consist of only one run (cf. Definition 5.5). Consequently, we have to store at most $\prod_{i=1}^{\mathsf{run}(P)} \left(\lfloor \frac{\mathsf{run}(F(i)}{2} \rfloor + 1 \right)$ many (κ, F)-matchings, but this number is still too large to show our desired runtime bounds. However, it suffices to distinguish between $\lfloor \mathsf{run}(F(i))/2 \rfloor$ many vales in $F(i)$ with $i \in [\mathsf{run}(P) - 1]$. This is achieved by not distinguishing between the first and the last vale in $F(\mathsf{ri}(i))$ for $i < \mathsf{run}(P)$. We only briefly mention that this is correct due to the Conditions (C5) and (C6); a formal proof will follow with Lemma 5.9 in Section 5.1.3. For $i = \mathsf{run}(P)$, the last run in P, we still consider all vales occurring in $F(\mathsf{run}(P))$.

Recall that our goal is to assign a position in the array X_κ^F to every (κ, F)-matching $\vec{x}$. For every one of the $\mathsf{run}(P)$ values of the (κ, F)-matching there are at most $\lfloor \mathsf{run}(F(\mathsf{ri}(i))/2 \rfloor$ vales to be distinguished, except for the last one where we distinguish between $\lfloor \mathsf{run}(F(\mathsf{run}(P)))/2 \rfloor + 1$ vales. Thus, it is natural to use a mixed radix numeral system with bases $b_1 = \lfloor \mathsf{run}(F(1)/2 \rfloor$, $b_2 = \lfloor \mathsf{run}(F(2)/2 \rfloor, \ldots, b_{\mathsf{run}(P)-1} = \lfloor \mathsf{run}(F(\mathsf{run}(P)-1)/2 \rfloor$ and $b_{\mathsf{run}(P)}$ is equal to the number of vales in $F(\mathsf{run}(P))$. Let `Index` be the function that assigns a position in the array to each (κ, F)-matching $\vec{x} = (x_1, \ldots, x_{\mathsf{run}(P)})$:

$$\texttt{Index}\left(x_1, \ldots, x_{\mathsf{run}(P)}\right) = 1 + \sum_{i=1}^{\mathsf{run}(P)} \left(\mathsf{vi}(x_i) - 1 \mod b_i\right) \cdot \prod_{j=1}^{i-1} b_j.$$

The mod operator is required since for $x \in F(i)$, $\mathsf{vi}(x) \in [b_i + 1]$ – as explained above.

	$\mathtt{Index}(.,.,.) = 1$	$\mathtt{Index}(.,.,.) = 2$
X_1^F	$(0,2,0)$	ϵ
X_2^F	$(8,2,0)$	$(4,2,0)$
X_3^F	$(6,2,0)$	$(11,2,0)$
X_4^F	$(6,2,9)$	ϵ

Table 5.1: The arrays $X_1^F, \ldots, X_4^F$ for our running example (cf. Figure 5.2)

Example 5.7. Let us discuss what the `Index` function looks like for our running example P_{ex} and T_{ex} (cf. Figure 5.2). The subsequence $F(1)$ contains four runs. Thus, $b_1 = 2$. Since both $F(2)$ and $F(3)$ contain two runs, $b_2 = b_3 = 1$. Consequently, in our running example, X_κ^F contains at most two elements for every $\kappa \in [k]$. For example, $\mathtt{Index}(8,3,10) = 1$, $\mathtt{Index}(6,3,10) = 1$ and $\mathtt{Index}(11,3,10) = 2$. $\dashv$

From the definition of the `Index`-function, it follows immediately that the length of our array is $\prod_{i=1}^{\mathsf{run}(P)} b_i$. We will show in Lemma 5.12 that $\prod_{i=1}^{\mathsf{run}(P)} b_i = \mathcal{O}\left(1.2611^{\mathsf{run}(T)}\right)$. At this point, we see the huge advantage of this array data structure over the set data structure in the simple algorithm: the set X_κ^F has a potential size of $n^{\mathsf{run}P}$ – too large for an fpt algorithm.

This concludes the description of the array data structure. Let us now – once again – return to our running example and see how this would be dealt with by the alternating run algorithm.

Example 5.8. Let us demonstrate how the alternating run algorithm works. As before, consider T_{ex}, P_{ex} and the matching function F as represented in Figure 5.2. We already know from the last example that X_κ^F has size 2, i.e., the `Index` function has range $\{1,2\}$. We start with $X_0^F = \{(0,0,0)\}$. Refer to Table 5.1 for an overview. For the element 1 in P the only representative element is 2. Since $\mathtt{Index}(0,2,0) = 1$, we store this $(1,F)$-matching at position 1 in X_1^F. Position 2 remains empty (symbolized by ϵ). For the element 2, we have more representative elements: $\mathtt{Rep}((0,2,0),2,F) = \{4,8\}$. Note that 3 is not a representative element since there is no larger element to its right in $F(\mathsf{ri}(2)) = F(1)$ (cf. (C6)). Since $\mathtt{Index}(8,2,0) = 1$ and $\mathtt{Index}(4,2,0) = 2$, both $(2,F)$-matchings are stored in X_2^F. For placing the element 3, observe that 3 is the largest element in its run in P. Thus, Condition (C5) applies. We obtain $\mathtt{Rep}((8,2,0),3,F) = \min\{11,12\} = \{11\}$ as well as $\mathtt{Rep}((4,2,0),3,F) = \min\{7,6\} = \{6\}$. Thus, we have two $(3,F)$-matchings to store in X_3^F: $(11,2,0)$ and $(6,2,0)$ with $\mathtt{Index}(11,2,0) = 2$ and $\mathtt{Index}(6,2,0) = 1$. Finally, we have to place the element 4. The $(3,F)$-matching $(11,2,0)$ does not lead to a matching since $\mathtt{Rep}((11,2,0),4,F) = \emptyset$. However, $\mathtt{Rep}((6,2,0),3,F) = \{9\}$. Thus, X_4^F contains the $(4,F)$-matching $(6,2,9)$. This $(4,F)$-matching corresponds to the matching $\{2 \mapsto 4, 3 \mapsto 6, 1 \mapsto 2, 4 \mapsto 9\}$. $\dashv$

Finally, it only remains to explain the `GetMatching` procedure.

From Lemma 5.5 we know that if there is an element in X_k^F, a matching from P into T that is compatible with F exists. However, we have not yet shown how a matching can be constructed from an element in X_k^F. This is what the `GetMatching` procedure does: it extracts an actual

Procedure GetMatching($X_1^F, \ldots, X_k^F$)

input : k arrays $X_1^F, X_2^F, \ldots, X_k^F$ generated by Algorithm 2
output: M, a matching of P into T that is compatible with F

for $\kappa \leftarrow k \ldots 1$ **do**
  if $\kappa = k$ **then**
    $\vec{x} \leftarrow$ some element in X_k^F
  else
    $\vec{x} \leftarrow$ some element $\vec{y}$ in X_κ^F with $x_i = y_i$ for all $i \neq \mathsf{ri}(\kappa)$
  $M(\kappa) \leftarrow x_{\mathsf{ri}(\kappa)}$
return $M = (M(1), M(2), \ldots, M(k))$

matching $M : [k] \to [n]$ out of the arrays $X_1^F, \ldots, X_k^F$. We construct M recursively: First, we pick some element $\vec{x} \in X_k^F$ and set $M(k) := x_{\mathsf{ri}(k)}$. Now, suppose the matching has been determined for $\kappa \in [k]$ and $M(\kappa) = x_{\mathsf{ri}(\kappa)}$ for some $\vec{x} \in X_\kappa^F$. Then there must exist an element $\vec{y} \in X_{\kappa-1}^F$ that has led to the element $\vec{x} \in X_\kappa^F$, i.e., $\vec{y}$ differs from $\vec{x}$ only at the $\mathsf{ri}(\kappa)$-th element. We define $M(\kappa - 1) := y_{\mathsf{ri}(\kappa-1)}$. This defines the function $M : [k] \to [n]$. It can easily be seen with the help of Lemma 5.4 that the function M returned by the `GetMatching` procedure is indeed a matching of P into T that is compatible with F.

This concludes our description of the alternating run algorithm. We would like to remark that this description omits two minor details necessary for obtaining the polynomial factor $\mathcal{O}(n \cdot k)$ of the desired runtime. The one detail concerns the calculation of the `Index` function. The second details concerns how data is stored in the array. These details are described in the proof of the runtime, Proposition 5.13.

5.1.3 Correctness

We start by providing the proof of Lemma 5.3, which states that for every matching M there exists a matching function F such that M is compatible with F.

Proof of Lemma 5.3. Given a matching M from P to T, we will construct a matching function F such that M is compatible with F. In order to describe F, it is enough to determine the first (=leftmost) element $l_{F(i)}$ of every $F(i)$, where $i \in [\mathsf{run}(P)]$. In order to specify the last (=rightmost) element $r_{F(i)}$ of $F(i)$ for $i \in [\mathsf{run}(P)]$, we simply need to apply the properties (P3) and (P4): $r_{F(i)}$ is either the last element in T or the leftmost valley (peak) in $F(i+1)$ in case that the i-th run is a run up (down). Clearly, $l_{F(1)} = T(1)$, the first element in T – cf. (P3). When determining $l_{F(i)}$, let $l_{P,i}$ be the first element in the i-th run in P and $r_{P,i}$ be the last element in the i-th run in P. If the i-th run is a run up (down), $l_{F(i)}$ is the right-most element in T lying to the left of $M(l_{P,i})$ and following a valley (peak). This construction guarantees that F is a matching function.

In order to prove that M is compatible with F, we need to show for all $i \in [\mathsf{run}(P)]$ that $l_{F(i)} \preceq_T M(l_{P,i})$ and $M(r_{P,i}) \preceq_T r_{F(i)}$. The first statement holds by construction. For $i = \mathsf{run}(P)$, the second statement clearly also holds by construction. Let $i \in [\mathsf{run}(P) - 1]$. Let

$\diamond$ and $\circ$ in the same run	$\diamond$ and $\circ$ not in the same run; $\circ$ in a run up	$\diamond$ and $\circ$ not in the same run; $\circ$ in a run down

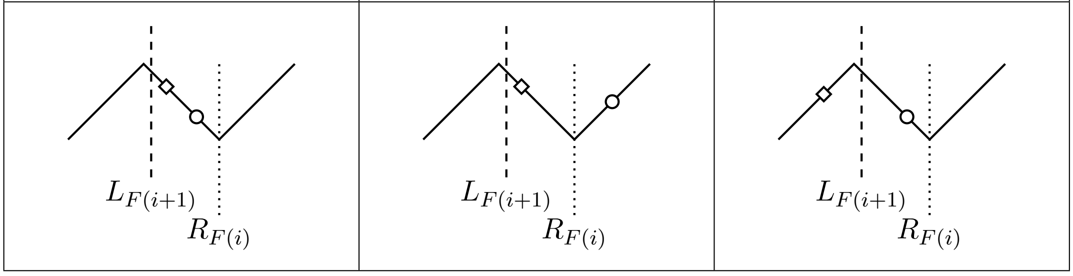


Figure 5.4: Three cases that have to be distinguished in the proof of Lemma 5.3 when showing that $M(r_{P,i}) \preceq_T r_{F(i)}$ for all $i \in [\mathsf{run}(P) - 1]$ under the assumption that the i-th run in P is a run up. The element $M(r_{P,i})$ is represented by a $\diamond$ and the element $M(l_{P,i+1})$ by a $\circ$.

us assume that the i-th run is a run up – the proof for runs down is analogous. We distinguish between the following cases that are depicted in Figure 5.4:

- $M(r_{P,i})$ and $M(l_{P,i+1})$ lie in the same run in T. Since we have assumed that the i-th run in P is a run up, $r_{P,i}$ is a peak in P. Hence, this case is only possible if $M(r_{P,i})$ is in a run down in T and $r_{P,i} > l_{P,i+1}$. Thus, $l_{F(i+1)}$ is the first element in this run, which implies that $r_{F(i)}$ is the last element of this run and thus $M(r_{P,i}) \preceq_T r_{F(i)}$.
- $M(r_{P,i})$ and $M(l_{P,i+1})$ do not lie in the same run in T and $M(l_{P,i+1})$ is in a run up in T. In this case, $r_{F(i)}$ is the last element in the run down preceding this run and thus it clearly holds that $M(r_{P,i}) \preceq_T r_{F(i)}$.
- $M(r_{P,i})$ and $M(l_{P,i+1})$ do not lie in the same run in T and $M(l_{P,i+1})$ is in a run down in T. In this case, $r_{F(i)}$ is the last element in this run and again it clearly holds that $M(r_{P,i}) \preceq_T r_{F(i)}$.

$\square$

Example 5.9. Constructing F as described in the proof of Lemma 5.3 for the matching $4\,6\,2\,9$ of P_{ex} into T_{ex} yields the matching function represented in Figure 5.2. $\dashv$

Next, we prove Lemma 5.4. This lemma states that a function $M{:}[k] \to [n]$ is a matching of P into T compatible with F if and only if for every $\kappa \in [k]$:

1. $M(\kappa) \in F(\mathsf{ri}(\kappa))$,
2. $M(\kappa) > M(\kappa - 1)$ and
3. if $\mathsf{pre}(\kappa)$ exists, then $\mathsf{pre}(\kappa) \prec_P \kappa$ if and only if $M(\mathsf{pre}(\kappa)) \prec_T M(\kappa)$, i.e., if κ is contained in a run up (down), then $M(\kappa)$ is right (left) of $M(\mathsf{pre}(\kappa))$.

Proof of Lemma 5.4. Let $M{:}[k] \rightarrow [n]$ be a matching of P into T that is compatible with F. Recall Definition 2.1 which states that M has to be a monotonically increasing function. This implies the second condition. Moreover, the sequence $M(P) = M(P(1)), M(P(2)), \ldots,$ $M(P(k))$ has to be a subsequence of T. This means nothing else than

$$M(P(1)) \prec_T M(P(2)) \prec_T \ldots \prec_T M(P(k)).$$

In particular it must hold that $M(i) \prec_T M(j)$, where i and j are two neighbouring elements in the same run in P with $i \prec_P j$. This implies the third condition. Finally, the first condition follows directly from the definition of compatibility (Definition 5.3).

Let $M{:}[k] \rightarrow [n]$ be a function fulfilling the three conditions stated above. The second condition implies that M is monotonically increasing. In order to show that M is indeed a matching of P into T, we have to show that $M(P) = M(P(1)), M(P(2)), \ldots, M(P(k))$ is a subsequence of T. In other words, we have to show that for all $i \in [k-1]$ it holds that $M(P(i)) \prec_T M(P(i+1))$. We distinguish three cases:

- The elements $P(i)$ and $P(i+1)$ lie in the same run in P. Thus, for the case of a run up (down) we have $P(i) = \mathsf{pre}(P(i+1))$ $(P(i+1) = \mathsf{pre}(P(i)))$. With $\kappa = P(i+1)$ $(\kappa = P(i))$ it follows from the third condition that $M(P(i)) \prec_T M(P(i+1))$ (in both cases).

- The elements $P(i)$ and $P(i+1)$ do not lie in the same run in P and $M(P(i))$ and $M(P(i+1))$ do not lie in the same run in T. If $P(i)$ lies in the j-th run in P, the first condition implies that $M(P(i))$ lies in $F(j)$ and that $M(P(i+1))$ lies in $F(j+1)$ in T. Then property (P4) of matching functions (the leftmost run of $F(j+1)$ is the rightmost run of $F(j)$) implies that $M(P(i))$ lies to the left of $M(P(i+1))$ in T.

- The elements $P(i)$ and $P(i+1)$ do not lie in the same run in P but $M(P(i))$ and $M(P(i+1))$ lie in the same run in T. By the definition of matching functions and since it holds that $M(\kappa) \in F(\mathsf{ri}(\kappa))$ for all $\kappa \in [k]$, this can only be possible if $M(P(i))$ is in the last run of $F(j)$ and $M(P(i+1))$ is in the first run of $F(j+1)$ for some $j \in [\mathsf{run}(P)]$. Thus, if $P(i)$ lies in a run up (down) in P both $M(P(i))$ and $M(P(i+1))$ are contained in a run down (up) in T. On the other hand, if $P(i)$ is in a run up (down) it must be a peak (valley) and thus it holds that $P(i) > P(i+1)$ $(P(i) < P(i+1))$. The second condition then ensures that $M(P(i)) > M(P(i+1))$ $(M(P(i)) < M(P(i+1)))$, which implies that $M(P(i))$ lies to the left of $M(P(i+1))$ in T.

The function M is thus a matching of P into T additionally fulfilling that $M(\kappa) \in F(\mathsf{ri}(\kappa))$ which means that M is a matching compatible with F. $\square$

Lemma 5.5 states that in Algorithm 1, $\vec{x} \in X_\kappa^F$ is a (κ, F)-matching. This can be shown as follows:

Proof of Lemma 5.5. We prove this statement by induction over κ. For $\kappa = 1$ this is easy: An element $\vec{x} \in X_1^F$ looks as follows: $x_i = 0$ for all $i \neq \mathsf{ri}(1)$ and $x_{\mathsf{ri}(1)}$ is equal to some $j \in F(\mathsf{ri}(1))$. Thus, the function $M : [1] \rightarrow [n]$ with $M(1) = j$ is clearly a $(1, F)$-matching.

Now suppose we have proven the statement of Lemma 5.5 for $\kappa - 1$ and we want to prove it for κ. If $\vec{x} \in X^F_\kappa$, then there must exist an element $\vec{y} \in X^F_{\kappa-1}$ and an element $\nu \in [n]$ such that $\vec{x} = (y_1, \ldots, y_{\mathsf{ri}(\kappa)-1}, \nu, y_{\mathsf{ri}(\kappa)+1}, \ldots, y_{\mathsf{run}(P)})$ (see lines 5 to 8 in Algorithm 1). This element ν may not be any arbitrary element, it must fulfill the following conditions (see Algorithm 1, Line 6): $\nu \in F(\mathsf{ri}(\kappa))$, $\nu > x_{\mathsf{ri}(\kappa-1))}$ and $\mathsf{pre}(\kappa) \prec_P \kappa$ if and only if $x_{\mathsf{ri}(\mathsf{pre}(\kappa))} \prec_T \nu$. Since $\vec{y} \in X^F_{\kappa-1}$ it is a $(\kappa-1, F)$-matching and thus there exists a function $M : [\kappa-1] \to [n]$ that is a matching of $P|_{[\kappa-1]}$ into T that is compatible with F and for which it additionally holds that for every $y_i \neq 0$, $M(\max\{\kappa' \leq \kappa - 1 : \mathsf{ri}(\kappa') = i\}) = y_i$.

We now define a function $\tilde{M} : [\kappa] \to [n]$ as follows: $\tilde{M}(i) = M(i)$ for all $i \in [\kappa-1]$ and $\tilde{M}(\kappa) = \nu$. We will see that this function $\tilde{M}$ is a witness for the fact that $\vec{x}$ is a (κ, F)-matching. For this purpose we have to check that the three conditions in Lemma 5.4 are fulfilled for every $i \in [\kappa]$. For $i < \kappa$ these conditions are necessarily fulfilled since we then have $\tilde{M}(i) = M(i)$ and M is a matching of $P|_{[\kappa-1]}$ into T that is compatible with F. For $i = \kappa$, i.e., $\tilde{M}(i) = \nu$, these conditions are exactly those stated above that must be fulfilled by the element $\nu \in [n]$. The last condition in Definition 5.4, namely that for every $x_i \neq 0$, $\tilde{M}(\max\{\kappa' \leq \kappa : \mathsf{ri}(\kappa') = i\}) = x_i$, is fulfilled since M is a witness for the fact that $\vec{y}$ is a $(\kappa-1, F)$-matching and since we defined $\tilde{M}(\kappa)$ to be equal to $\nu = x_{\mathsf{ri}(\kappa)}$. Thus, $\vec{x}$ is a (κ, F)-matching. □

The next lemma shows that only considering elements returned by the `Rep` procedure is sound.

Definition 5.6. *Let F be a matching function and $\vec{x} = (x_1, x_2, \ldots, x_{\mathsf{run}(P)})$ be a (κ, F)-matching for some $\kappa \in [k]$. A matching M (κ, F)-*extends *$\vec{x}$ if M is compatible with F and if for every $x_i \neq 0$, $M(\max\{\kappa' \leq \kappa : \mathsf{ri}(\kappa') = i\}) = x_i$, i.e., M maps the largest element $\leq \kappa$ in the i-th run of P to the i-th element of $\vec{x}$.*

Definition 5.7. *Let $\vec{x} = (x_1, \ldots, x_{\mathsf{run}(P)})$. In the following, we write $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu)$ instead of $(x_1, \ldots, x_{\mathsf{ri}(\kappa)-1}, \nu, x_{\mathsf{ri}(\kappa)+1}, \ldots, x_{\mathsf{run}(P)})$.*

Lemma 5.7. *Let $\kappa \in [k]$ and $\vec{x} \in X^F_\kappa$. If there exists a matching M that (κ, F)-extends $\vec{x}$, then there exist an element $\nu \in$ `Rep`$(\vec{x}, \kappa+1, F)$ and a matching $\tilde{M}$ that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu)$.*

Proof. Let us first explicitly show how to pick the element ν. Then we will prove that it indeed holds that ν is in `Rep`$(\vec{x}, \kappa+1, F)$. We define $\tilde{M}$ as follows: $\tilde{M}(\kappa+1) := \nu$ and $\tilde{M}(i) := M(i)$ otherwise. Finally, we will see that $\tilde{M}$ is a matching that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu)$.

In order to increase legibility, let $i \in [k]$ be such that $P(i) = \kappa + 1$. Let us then consider the set S consisting of all elements in T that lie strictly to the right of $M(P(i-1))$ and strictly to the left of $M(P(i+1))$, that are contained in $F(\mathsf{ri}(\kappa+1))$ and that are larger than $M(\kappa) = x_{\mathsf{ri}(\kappa)}$. Thus,

$$S := \{j \in [n] : M(P(i-1)) \prec_T j \prec_T M(P(i+1))\} \cap F(\mathsf{ri}(\kappa+1)) \cap [M(\kappa)+1, n].$$

This set is never empty: Especially, $M(\kappa+1)$ is contained in S since M is a matching that (κ, F)-extends $\vec{x}$. We now define $\nu := \min(S)$.

We have to check that it indeed holds that $\nu \in$ `Rep`$(\vec{x}, \kappa+1, F)$. We refer the reader to the definition of `Rep`$(\vec{x}, \kappa+1, F)$ on page 44.

- (C1) is fulfilled by construction of S.
- (C2) is fulfilled since $\nu > M(\kappa - 1) = x_{\mathsf{ri}(\kappa-1)}$.
- (C3) is fulfilled: ν is a valley in the subsequence of T consisting of elements larger than $M(\kappa)$ by construction of S.
- (C4) If the run predecessor of $\kappa+1$ exists and $\kappa+1$ lies in a run up (down), $\mathsf{pre}(\kappa+1) = P(i-1)$ ($\mathsf{pre}(\kappa+1) = P(i+1)$). Moreover, note that $M(\mathsf{pre}(\kappa+1)) = x_{\mathsf{ri}(\kappa+1)}$ since M (κ, F)-extends $\vec{x}$. Since $S \subseteq \{j \in [n] : M(P(i-1)) \prec_T j \prec_T M(P(i+1))\}$, it is guaranteed that ν lies on the correct side of $x_{\mathsf{ri}(\kappa+1)}$.
- (C5) In case $\kappa+1$ is the largest element in its run in P, there is only a single element in $\mathtt{Rep}(\vec{x}, \kappa+1, F)$ which is exactly ν.
- (C6) In case $\kappa+1$ is not the largest element in its run in P and $\kappa+1$ lies in a run up (down), the element $M(P(i+1))$ ($M(P(i-1))$) is an element larger than ν that lies to the right (left) of ν in $F(\mathsf{ri}(\kappa+1))$ since M is compatible with F.

Now let us show that $\tilde{M}$ as defined above is a matching that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu)$. First we need to show that the function $\tilde{M}$ is a matching of P into T that is compatible with F. Here Lemma 5.4 comes in handy since it tells us that we only have to check the following three conditions for all $j \in [k]$:

1. $\tilde{M}(j) \in F(\mathsf{ri}(j))$: For $j = \kappa+1$ this holds by construction of ν and for $j \neq \kappa+1$ this holds since we then have $\tilde{M}(j) = M(j)$ and M is a matching that is compatible with F.
2. $\tilde{M}(j+1) > \tilde{M}(j)$ for $j \neq k$: For $j \notin \{\kappa, \kappa+1\}$ this again holds since M is a matching.

   $j = \kappa$: By the construction of S, $\tilde{M}(\kappa+1) = \nu > M(\kappa) = \tilde{M}(\kappa)$.

   $j = \kappa+1$: Again by the construction of S we know that $\nu \leq M(\kappa+1)$. Since M is a matching $M(\kappa+1) < M(\kappa+2) = \tilde{M}(\kappa+2)$ it follows that $\nu = \tilde{M}(\kappa+1) < \tilde{M}(\kappa+2)$.
3. If $\mathsf{pre}(j)$ exists, then $\mathsf{pre}(j) \prec_P j$ if and only if $\tilde{M}(\mathsf{pre}(j)) \prec_T \tilde{M}(j)$: Since M is a matching, we only have to check this condition for $\kappa+1$ and its run predecessor $\mathsf{pre}(\kappa+1)$ as well as for $\kappa+1$ and κ', the next largest element in the same run in P (we could call this element the run successor of $\kappa+1$), i.e., $\mathsf{pre}(\kappa') = \kappa+1$. If $\kappa+1$ lies in a run up (down), we have $\mathsf{pre}(\kappa+1) = P(i-1)$ and $\kappa' = P(i+1)$ ($\mathsf{pre}(\kappa+1) = P(i+1)$ and $\kappa' = P(i-1)$). By construction of S we have that $M(P(i-1)) = \tilde{M}(P(i-1)) \prec_T \nu = \tilde{M}(\kappa+1) \prec_T \tilde{M}(P(i+1)) = M(P(i+1))$ and thus this condition is also fulfilled.

In order to show that $\tilde{M}$ $(\kappa+1, F)$-extends $\vec{y} := \vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu)$ it remains to show that for every $y_i \neq 0$, $\tilde{M}(\max\{\kappa' \leq \kappa : \mathsf{ri}(\kappa') = i\}) = y_i$. For $i \neq \mathsf{ri}(\kappa+1)$ this follows from the fact that $y_i = x_i$ and that M is a matching that (κ, F)-extends $\vec{x}$. For $i = \mathsf{ri}(\kappa+1)$ this hold by definition of $\tilde{M}$: we have $y_i = \nu$ and $\tilde{M}(\max\{\kappa' \leq \kappa+1 : \kappa' \text{ is in the same run as } \kappa+1\}) = \tilde{M}(\kappa+1) = \nu$. $\square$

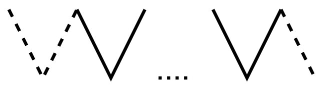
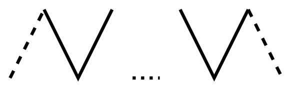
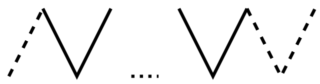
	possible for $i = 1$	possible for $i \in [1, \mathsf{run}(P) - 1]$
the i-th run in P is a run up	 b_1 = number of vales in $F(1) - 1$	 b_i = number of vales in $F(i) - 1$
the i-th run in P is a run down	 b_1 = number of vales in $F(1) - 1$	 b_i = number of vales in $F(i)$

Figure 5.5: Possible shapes that $F(i)$ can have in T, where $i \neq \mathsf{run}(P)$. Runs that are drawn with dashed lines indicate that elements x lying in these runs fulfil $\mathsf{vi}(x) \equiv 1 \mod b_i$.

It remains to prove that the use of the array data structure and in particular the `Index` function do not cause that relevant (κ, F)-matchings are discarded. This is done by the following two lemmas.

Lemma 5.8. *Let $\vec{x}, \vec{y}$ be two (κ, F)-matchings, where $\kappa \in [k]$ and F is a matching function. If $\mathtt{Index}(\vec{x}) = \mathtt{Index}(\vec{y})$, then for all $i \in [\mathsf{run}(P)]$ it holds that*

- *x_i and y_i lie in the same vale in T or*
- *the largest element in the i-th run in P is smaller or equal to κ.*

Proof. From the definition of the `Index` function it is clear that $\mathtt{Index}(\vec{x}) = \mathtt{Index}(\vec{y})$ implies that $\mathsf{vi}(x_i) \equiv \mathsf{vi}(y_i) \mod b_i$ for all $i \in [\mathsf{run}(P)]$. Recall that for $i = \mathsf{run}(P)$, b_i corresponds exactly to the number of vales in $F(i)$ and thus $\mathsf{vi}(x_{\mathsf{run}(P)}) \equiv \mathsf{vi}(y_{\mathsf{run}(P)}) \mod b_i$ is only possible if $\mathsf{vi}(x_{\mathsf{run}(P)}) = \mathsf{vi}(y_{\mathsf{run}(P)})$ which means nothing else than that $x_{\mathsf{run}(P)}$ and $y_{\mathsf{run}(P)}$ lie in the same vale in T.

For the case that $i \neq \mathsf{run}(P)$, this is not always as simple. Consider the four possible shapes that $F(i)$ can have, as depicted in Figure 5.5. Let us first take a look at the case that the i-th run in P is a run up. Here, $\mathsf{vi}(x_i) \equiv \mathsf{vi}(y_i) \mod b_i$ is possible if x_i and y_i lie in the same vale in T or if x_i lies in the first vale in $F(i)$ and y_i lies in the last run in $F(i)$ (or vice-versa). Now recall the definition of the `Rep` procedure: an element in the last run (which is always a run down) may only be chosen for the largest element in its run in P (Condition (C6)). This means that the largest element in the i-th run in P must be smaller or equal to κ. Now let us consider the case that the i-th run in P is a run down. Here, if x_i and y_i do not lie in the same vale in T, $\mathsf{vi}(x_i) \equiv \mathsf{vi}(y_i) \mod b_i$ is only possible for $i = 1$ and if T starts with a run up: x_i has to then lie in this first run of T and y_i in the last vale of $F(1)$ (or vice-versa). Again, because of Condition

(C6), this is only possible for the largest element in its run in P. Thus, we can again conclude that the largest element in the i-th run in P must be smaller or equal to κ. $\square$

Lemma 5.9. *Let $\vec{x}, \vec{y}$ be two (κ, F)-matchings, where $\kappa \in [k]$ and F is a matching function. In addition to that, let $\nu_x \in \mathtt{Rep}(\vec{x}, \kappa+1, F)$ and $\nu_y \in \mathtt{Rep}(\vec{y}, \kappa+1, F)$. If*

$$\mathtt{Index}(\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)) = \mathtt{Index}(\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y))$$

and $\nu_y \leq \nu_x$ the following holds: if there exists a matching that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)$, then there exists a matching that $(\kappa+1, F)$-extends $\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y)$. Thus, the alternating run algorithm only has to keep track of the $(\kappa+1, F)$-matching $\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y)$.

Proof. Let M_x be a matching of P into T that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)$. We shall construct a function $M_y : [k] \to [n]$ and show that it is a matching that $(\kappa+1, F)$-extends $\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y)$.

Since $\vec{y}$ is a (κ, F)-matching (Recall Definition 5.4) there exists a partial matching $M : [\kappa] \to [n]$ of $P|_{[\kappa]}$ into T for which it additionally holds that for every $y_i \neq 0$, $M(\max\{\kappa' \leq \kappa : \mathsf{ri}(\kappa') = i\}) = y_i$. We define the function M_y as follows:

$$M_y(i) = \begin{cases} M(i), & \text{for } i \in [\kappa] \\ \nu_y, & \text{for } i = \kappa+1 \\ M_x(i), & \text{for } i \in [\kappa+2, k] \end{cases}$$

We now need to show that M_y is indeed a matching that $(\kappa+1, F)$-extends $\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y)$. As in the proof of Lemma 5.7, we shall use Lemma 5.4 to show that M_y is a matching that is compatible with F. We have to check the following three conditions for all $j \in [k]$:

1. $M_y(j) \in F(\mathsf{ri}(j))$: For $j = \kappa+1$ this holds since $\nu_y \in \mathtt{Rep}(\vec{y}, \kappa+1, F)$ (Condition (C1)) and for $j \neq \kappa+1$ this holds since M_x and M are matchings that are compatible with F.

2. $M_y(j+1) > M_y(j)$ for $j \neq k$: For $j \notin \{\kappa, \kappa+1\}$ this again holds since M_x and M are matchings.

   a) $M_y(\kappa+1) > M_y(\kappa)$ or equivalently $\nu_y > M(\kappa) = y_{\mathsf{ri}(\kappa)}$: This holds since $\nu_y \in \mathtt{Rep}(\vec{y}, \kappa+1, F)$ (Condition (C2)).

   b) $M_y(\kappa+2) > M_y(\kappa+1)$ or equivalently $M_x(\kappa+2) > \nu_y$: Since M_x is a matching that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)$ it has to hold that $M_x(\kappa+2) > M_x(\kappa+1) = \nu_x$. Since we have $\nu_y \leq \nu_x$, this condition is fulfilled.

3. If $\mathsf{pre}(j)$ exists, then $\mathsf{pre}(j) \prec_P j$ if and only if $M_y(\mathsf{pre}(j)) \prec_T M_y(j)$: Since M_x and M are matchings, this condition is fulfilled for all $j \in [k]$ such that both $j < \kappa+1$ and $\mathsf{pre}(j) < \kappa+1$ or such that both $j > \kappa+1$ and $\mathsf{pre}(j) > \kappa+1$. Thus, we only have to check this condition for $j = \kappa+1$ and for all $\kappa' \in [\kappa+2, k]$ that satisfy $\mathsf{pre}(\kappa') \leq \kappa+1$. Let K be the set of all such κ'. Observe that such a κ' is the smallest element in the $\mathsf{ri}(\kappa')$-th run in P that is strictly larger than $\kappa+1$. This means that $\mathsf{pre}(\kappa')$, if it exists, is the largest element in the $\mathsf{ri}(\kappa')$-th run in P that is smaller or equal to $\kappa+1$. We only consider

the case that j is contained in a run up – the proof for the case that j lies in a run down works analogously. We have to check the condition for the following three situations:

a) $j = \kappa + 1$: If $\mathsf{pre}(\kappa+1)$ exists it has to hold that $M_y(\mathsf{pre}(\kappa+1)) = y_{\mathsf{ri}(\kappa+1)} \prec_T \nu_y$. This condition is fulfilled since $\nu_y \in \mathtt{Rep}(\vec{y}, \kappa+1, F)$ (Condition (C4)).

b) $j = \kappa' \in K$ such that $\mathsf{pre}(\kappa') = \kappa + 1$: If this element κ' exists we have to show that $\nu_y \prec_T M_y(\kappa') = M_x(\kappa')$. Since $\kappa + 1$ is not the largest element in its run in P, we know from Lemma 5.8 that ν_x and ν_y lie in the same vale in T. Moreover we know that $\nu_x \geq \nu_y$ – but what does this imply for the right-left order of ν_x and ν_y within this vale? Two cases may occur: ν_x may lie in the run up or in the run down of this vale. If ν_x lies in the run up, then it has to hold that $\nu_y \prec_T \nu_x$. Since M_x is a matching, it has to hold that $\nu_x = M_x(\kappa+1) \prec_T M_x(\kappa')$ and thus $\nu_y \prec_T M_x(\kappa')$. If ν_x lies in the run down, $\nu_x \prec_T \nu_y$ and all elements between ν_x and ν_y in T are smaller than v_x. This implies that $M_x(\kappa')$ which is larger than ν_x and lies to the right of ν_x also has to lie to the right of ν_y in T.

c) $j = \kappa' \in K$ with $\mathsf{pre}(\kappa') < \kappa + 1$: We need to show that $y_{\mathsf{ri}(\mathsf{pre}(\kappa'))} = y_{\mathsf{ri}(\kappa')} = M(\mathsf{pre}(\kappa')) \prec_T M_x(\kappa')$. Since M_x is a matching that $(\kappa+1, F)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)$, we know that $M_x(\mathsf{pre}(\kappa')) = x_{\mathsf{ri}(\kappa')}$ and that $x_{\mathsf{ri}(\kappa')} \prec_T M_x(\kappa')$. Moreover, since $\mathtt{Index}(\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu_x)) = \mathtt{Index}(\vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y))$ and $\mathsf{pre}(\kappa')$ is not the largest element in its run in P we know from Lemma 5.8 that $x_{\mathsf{ri}(\kappa')}$ and $y_{\mathsf{ri}(\kappa')}$ lie in the same vale in T. However, nothing is known about the relative positions of these two elements within this vale and we have to distinguish two cases. If $y_{\mathsf{ri}(\kappa')} \prec_T x_{\mathsf{ri}(\kappa')}$ the statement follows easily since $y_{\mathsf{ri}(\kappa')} \prec_T x_{\mathsf{ri}(\kappa')} \prec_T M_x(\kappa')$. If $x_{\mathsf{ri}(\kappa')} \prec_T y_{\mathsf{ri}(\kappa')}$ we have to collect a few more arguments in order to prove that the condition holds. By transitivity and the condition checked in Point 2. of this proof we know that $y_{\mathsf{ri}(\kappa')} < \nu_y = M_y(\kappa+1) < M_x(\kappa')$. Now note that the elements that lie in T between $x_{\mathsf{ri}(\kappa')}$ and $y_{\mathsf{ri}(\kappa')}$ are all smaller than $\max(x_{\mathsf{ri}(\kappa')}, y_{\mathsf{ri}(\kappa')})$ (since both are contained in the same vale). Thus, the element $M_x(\kappa')$ – that is to the right of $x_{\mathsf{ri}(\kappa')}$ and larger than $y_{\mathsf{ri}(\kappa')}$ – has to lie to the right of $y_{\mathsf{ri}(\kappa')}$. This is what we wanted to prove.

Let $\vec{y}' = \vec{y}(\mathsf{ri}(\kappa+1) \leftarrow \nu_y)$. It remains to show that for every $i \in [\mathsf{run}(P)]$ with $y'_i \neq 0$, $M_y(\max\{\kappa' \leq \kappa+1 : \mathsf{ri}(\kappa') = i\}) = y'_i$. This follows directly from the definition of $M_y(\kappa+1)$ and the fact that M is a witness for $\vec{y}$ being a (κ, F)-matching. □

Finally, we have gathered all necessary information to prove the correctness of the alternating run algorithm.

Proposition 5.10. *P can be matched into T if and only if X_k^F is non-empty for some matching function F.*

Proof. ($\Rightarrow$) If there is a matching of P into T, then there is at least one matching function F for which X_k^F is nonempty:
Since there exists a matching M, we know from Lemma 5.3 that there exists some matching function F such that M is compatible with F. Let us fix this F. We prove by induction over $\kappa \in$

$[k]$ that there is an $\vec{x} \in X^F_\kappa$ and a matching M_κ that (κ, F)-extends $\vec{x}$. For $\kappa = 1$ this is easy. Let ν be the valley in T that lies in the same vale as $M(1)$. It is clear that $\nu \in \texttt{Rep}((0,\dots,0), 1, F)$. Consequently, the tuple $\vec{x}$ with $x_i = 0$ for $i \neq 1$ and $x_{\mathsf{ri}(1)} = \nu$) is contained in X^F_1. Observe that M_1 being defined by $M_1(i) = M(i)$ for $i \neq 1$ and $M_1(1) = \nu$ is a matching that $(1, F)$-extends $\vec{x}$.

Now, let $\kappa \in [k]$ and assume that $\vec{x} \in X^F_\kappa$ and M_κ κ-extends $\vec{x}$. We show that there exist an $\vec{x}' \in X^F_{\kappa+1}$ and a $M_{\kappa+1}$ that $(\kappa+1)$-extends $\vec{x}'$. By Lemma 5.7, there exists a $\nu \in \texttt{Rep}(\vec{x}, \kappa+1, F)$ and a matching $M_{\kappa+1}$ that $(\kappa+1)$-extends $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu)$. At this point, we cannot be sure that $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu) \in X^F_{\kappa+1}$ since $X^F_{\kappa+1}$ may contain another (κ, F)-matching $\vec{y}$ with $\texttt{Index}(\vec{x}) = \texttt{Index}(\vec{y})$. However, this is only possible if $y_{\mathsf{ri}(\kappa+1)} \leq x_{\mathsf{ri}(\kappa+1)}$ (see Line 10 in Algorithm 2). By Lemma 5.9 we know that, in this case, there exists a matching that $(\kappa+1)$-extends $\vec{y}$. So, no matter whether $\vec{x}(\mathsf{ri}(\kappa+1) \leftarrow \nu) \in X^F_{\kappa+1}$ or not, we can conclude that there is an $\vec{x}' \in X^F_{\kappa+1}$ and a matching function $M_{\kappa+1}$ that $(\kappa+1)$-extends $\vec{x}'$. By induction, we have shown that $X^F_k \neq \emptyset$.
($\Leftarrow$) If there is a matching function F such that the corresponding X^F_k is non-empty, then a matching of P into T can be found: This is an immediate consequence of Corollary 5.6. $\square$

Finally, let us remark that the function M as returned by the procedure

$$\texttt{GetMatching}(X^F_1, \dots, X^F_k)$$

is indeed a matching, as can easily be seen with the help of Lemma 5.4: The first condition in the lemma is satisfied because of Condition (C1) for representative elements. The second condition holds because of Condition (C2). The third condition corresponds to Condition (C4). Note that(C3), (C5) and (C6) are only required for improving the runtime.

5.1.4 Runtime

We are now going to prove the promised fpt runtime bounds. First, we bound the number of matching functions.

Lemma 5.11. *There are less than $(\sqrt{2})^{\mathsf{run}(T)}$ functions from $[\mathsf{run}P]$ to subsequences of T that satisfy (P1) to (P4).*

Proof. A matching function F can be uniquely characterized by fixing the position of the first run up in every $F(i)$ for $i \in [\mathsf{run}(P)]$. This is because the last run of $F(i)$ is the first run of $F(i+1)$ for all $i \in [\mathsf{run}(P) - 1]$. Moreover the first run up in $F(1)$ is always the first run up in T. Thus, the number of matching functions is equal to the number of possibilities of picking $\mathsf{run}(P) - 1$ runs (for the first run in P no choice has to be made) among the at most $\lceil \mathsf{run}(T)/2 \rceil$ runs up in T. Hence, we obtain

$$\binom{\lceil \mathsf{run}(T)/2 \rceil}{\mathsf{run}(P) - 1} \leq 2^{\lceil \mathsf{run}(T)/2 \rceil - 1} < (\sqrt{2})^{\mathsf{run}(T)}.$$

The first inequality holds since $\binom{n}{k} < 2^{n-1}$ for all $n, k \in \mathbb{N}$ as can easily be proven by induction over n. $\square$

Now we bound the size of X_κ^F, which is the main step to achieve the $1.79^{\mathsf{run}(T)}$ runtime bound.

Lemma 5.12. *For any given matching function F and every $\kappa \in [k]$*

$$|X_\kappa^F| \leq 2 \cdot \prod_{i=1}^{run(P)} \frac{\mathsf{run}(F(i))}{2} \leq 1.6 \cdot 1.261071^{\mathsf{run}(T)}.$$

Proof. Recall that each (κ, F)-matching in X_κ^F has a position as determined by the function `Index`, defined by

$$\mathtt{Index}(x_1, \ldots, x_{\mathsf{run}(P)}) = 1 + \sum_{i=1}^{\mathsf{run}(P)} (\mathsf{vi}(x_i) \mod b_i) \cdot \prod_{j=1}^{i-1} b_j.$$

For $i \in [k-1]$, $b_i = \lfloor \mathsf{run}(F(i)/2 \rfloor$, and $b_{\mathsf{run}(P)} \leq \lfloor \mathsf{run}(F(\mathsf{run}(P))/2 \rfloor + 1$ since $b_{\mathsf{run}(P)}$ is equal to the number of vales in $F(\mathsf{run}(P))$ [1] The range of `Index` is $\left\{1, \ldots, \prod_{i=1}^{\mathsf{run}(P)} b_i\right\}$. Since the function `Index` determines the positions in the array X_κ^F, we obtain

$$|X_\kappa^F| = \prod_{i=1}^{\mathsf{run}(P)} b_i \leq \prod_{i=1}^{\mathsf{run}(P)-1} \left\lfloor \frac{\mathsf{run}(F(i))}{2} \right\rfloor \cdot \left(\left\lfloor \frac{\mathsf{run}(F(\mathsf{run}(P)))}{2} \right\rfloor + 1 \right)$$

and consequently

$$|X_\kappa^F| \leq 2 \cdot \prod_{i=1}^{\mathsf{run}(P)} \frac{\mathsf{run}(F(i))}{2}. \tag{5.1.1}$$

We want to bound X_κ^F and thus want to know when the product in Equation (5.1.1) is maximal. The maximum of this product has to be determined under the condition that

$$\sum_{i=1}^{\mathsf{run}(P)} \mathsf{run}(F(i)) = \mathsf{run}(T) + \mathsf{run}(P) - 1, \tag{5.1.2}$$

since two subsequent $F(i)$'s have one run in common (cf. Definition 5.2). The inequality of geometric and arithmetic means implies that the product in Equation (5.1.1) is maximal if all $\mathsf{run}(F(i))$ are equal, i.e., for every $i \in \mathsf{run}(P)$, $\mathsf{run}(F(i)) = \frac{\mathsf{run}(T)+\mathsf{run}(P)-1}{\mathsf{run}(P)}$. Therefore, X_κ^F has at most $2 \cdot \left(\frac{\mathsf{run}(T)+\mathsf{run}(P)-1}{2 \cdot \mathsf{run}(P)}\right)^{\mathsf{run}(P)}$ elements. To shorten the proof, we write in the following p for $\mathsf{run}(P)$ and t for $\mathsf{run}(T)$. Thus, we want to determine the maximum of the function

$$g(p) = \left(\frac{t+p-1}{2p}\right)^p$$

[1] The reason why we do not set $b_{\mathsf{run}(P)} = \lfloor \mathsf{run}(F(\mathsf{run}(P))/2 \rfloor$ is a rather technical one: $F(\mathsf{run}(P))$ may end with a run up if the last run in P is a run up and may end with a run down if the last run in P is a run down. This would lead to unwanted collisions concerning the `Index` function and consequently would prohibit the proof of Lemma 5.8.

(we omit the factor 2 for the calculation).

$$g'(p) = \frac{1}{p}\Bigg(2^{-p}\Big(\frac{p+t-1}{p}\Big)^{p-1}\cdot$$

$$\cdot\left((p+t-1)\log\left(\frac{p+t-1}{p}\right) - p\log(2) - t(1+\log(2)) + 1 + \log(2)\right)\Bigg) \stackrel{!}{=} 0$$

$$\implies (p+t-1)\left(\log\left(\frac{p+t-1}{p}\right) - \log(2)\right) - t + 1 = 0$$

$$\implies \log\left(\frac{p+t-1}{2p}\right) = \frac{t-1}{p+t-1}.$$

The solutions are:

$$\begin{aligned} p_1(t) &= (-1+t)/(-1+2e^{1+W_0(-1/(2e))}) \\ p_2(t) &= (-1+t)/(-1+2e^{1+W_{-1}(-1/(2e))}), \end{aligned}$$

where W_0 is the principal branch of the Lambert function (defined by $x = W(x)\cdot e^{W(x)}$) and W_{-1} its lower branch. It holds that

$$\begin{aligned} (-1+t)/3.311071 &\leq p_1(t) \leq (-1+t)/3.311070 \\ (-1+t)/-0.62663 &\leq p_2(t) \leq (-1+t)/-0.62664, \end{aligned}$$

The second solution $p_2(t)$ is negative and therefore of no interest to us. The first solution $p_1(t)$ is a local maximum as can be checked easily and yields

$$g(p_1) \leq \left(\frac{t+(-1+t)/3.311070-1}{2(-1+t)/3.311071}\right)^{(-1+t)/3.311070} \leq 0.80\cdot(1.261071)^t.$$

It therefore holds that $|X^F_\kappa| \leq 1.6\cdot 1.261071^{\mathsf{run}(T)}$. □

Proposition 5.13. *The runtime of the alternating run algorithm is $\mathcal{O}(1.784^{\mathsf{run}(T)}\cdot n\cdot k)$.*

Proof. The main structure of the algorithm is the following: for every matching function F and for every $\kappa \in [k]$ the array X^F_κ is computed. There are $(\sqrt{2})^{\mathsf{run}(T)}$ matching functions (Lemma 5.11). The maximal number of elements in X^F_κ is $1.6\cdot 1.2611^{\mathsf{run}(T)}$ (Lemma 5.12). Given a matching function and an element $\kappa \in [k]$, the algorithm has to execute Lines 6 to 11 for every $\vec{x} \in X^F_{\kappa-1}$. Once we have shown that the runtime of these lines is $\mathcal{O}(n)$, we obtain a total runtime of $\mathcal{O}\left((\sqrt{2})^{\mathsf{run}(T)}\cdot 1.2611^{\mathsf{run}(T)}\cdot k\cdot n\right) = \mathcal{O}(1.784^{\mathsf{run}(T)}\cdot k\cdot n)$.

So it remains to show that the runtime of the Lines 6 to 11 is $\mathcal{O}(n)$. First, observe that determining the set R with the help of the `Rep` procedure requires $\mathcal{O}(n)$ time. Second, for every element in R the Lines 8, 10 and 11 are executed. Since R only contains valleys (of some subsequence of T), its size is less than $\mathsf{run}(T)$. Assuming unit cost for arithmetic operations, computing `Index` requires $\mathcal{O}(\mathsf{run}(P))$ time. However, note that it is not necessary to repeat all calculations for $\texttt{Index}$ for every element ν in R. Indeed, for a fixed $\vec{x} \in X_\kappa^F$, the elements for which $\texttt{Index}$ is computed at Line 8 only differ at the $\mathsf{ri}(\kappa)$-th position. Assume that we have already computed $\texttt{Index}(\vec{x})$ for some $\vec{x}$. Computing $\texttt{Index}(\vec{y})$ for a $\vec{y}$ that is identical to $\vec{x}$ except at the $\mathsf{ri}(\kappa)$-th position can be done as follows:

$$\texttt{Index}(\vec{y}) = \texttt{Index}(\vec{x}) + \big(\mathsf{vi}(y_{\mathsf{ri}(\kappa)}) - \mathsf{vi}(x_{\mathsf{ri}(\kappa)}) \mod b_{\mathsf{ri}(\kappa)}\big) \cdot \prod_{j=1}^{\mathsf{ri}(\kappa)-1} b_j.$$

Consequently, Line 8 requires (amortized) constant time.

Checking the condition in Line 10 requires only constant time. However, Line 11 requires $\mathcal{O}(\mathsf{run}(P))$ time to write the (κ, F)-matching to its position in X_κ^F. This is too much time to obtain the desired runtime bound – we can only afford amortized $\mathcal{O}(n)$ time per $\vec{x} \in X_{\kappa-1}^F$. This can be achieved by executing Line 11 at most once per $\vec{x} \in X_{\kappa-1}^F$. Let $\nu \in R$ be the first element for which the condition at Line 10 is fulfilled. For this element Line 11 is executed and a pointer p' to the position $\texttt{Index}(\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu))$ is created. (Recall the $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu)$ notation from Definition 5.7.) If the condition at Line 10 is fulfilled for the same $\vec{x}$ and some other $\nu' \in R$, we do not execute Line 11. Instead we only store the pointer p' and the element ν'. This is sufficient information since two (κ, F)-matchings in Line 11 that originate from the same $\vec{x}$ are identical except for the $\mathsf{ri}(\kappa)$-th element. It might be that Line 11 is executed for some other element $\vec{y} \in X_{\kappa-1}^F$ and $\nu_y \in \texttt{Rep}(\vec{y}, \kappa, F)$ at a later point. It is then possible that a (κ, F)-matching $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu)$ is overwritten that has other (κ, F)-matchings $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu')$ pointing to it. However, this can only happen in the following situation: $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu')$ is (κ, F)-extendable only if $\vec{y}(\mathsf{ri}(\kappa) \leftarrow \nu')$ is (κ, F)-extendable. (It holds that $\texttt{Index}(\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu')) = \texttt{Index}(\vec{y}(\mathsf{ri}(\kappa) \leftarrow \nu'))$. Lemma 5.9 shows that if $\vec{x}(\mathsf{ri}(\kappa) \leftarrow \nu')$ is (κ, F)-extendable, then so is $\vec{y}(\mathsf{ri}(\kappa) \leftarrow \nu')$. Strictly speaking Lemma 5.9 is not applicable since it is not guaranteed that $\nu' \in \texttt{Rep}(\vec{y}, \kappa, F)$ because ν' might not be a valley in the corresponding subsequence of T (cf. Condition (C3)). However, all other conditions are satisfied and this suffices to prove Lemma 5.9.) Therefore, this modified array data structure is equivalent to the original data structure described in Section 5.1.2. Thus, we have shown that Lines 6 to 11 have a runtime of $\mathcal{O}(n)$, if we modify the array data structure to also allow for pointers. This concludes our proof. $\square$

We conclude this section about the runtime of the alternating run algorithm by proving that an even smaller constant than 1.784 can be expected. Indeed, the following holds:

Theorem 5.14. *Let R_n be the random variable counting the number of alternating runs in an n-permutation chosen uniformly at random amongst all n-permutations. Then for $n \geq 2$ we have:* $\mathbb{E}\left(1.784^{R_n}\right) = \mathcal{O}\left(1.515^n\right)$.

Proof. In the following, let $R_{n,i}$ denote the number of n-permutations with exactly i alternating runs. Then the mean of R_n is given as follows:

$$\mathbb{E}(R_n) = \sum_{i \geq 1} i \cdot \frac{R_{n,i}}{n!}.$$

By the law of the unconscious statistician (see any textbook on probability theory, e.g. [107]) we then have that:

$$\mathbb{E}\left(1.784^{R_n}\right) = \sum_{i\geq 1} 1.784^i \cdot \frac{R_{n,i}}{n!}.$$

Let $R_n(u) = \sum_{i\geq 1} R_{n,i}u^i$ denote the generating function of alternating runs in n-permutations. Then $\mathbb{E}\left(1.784^{R_n}\right)$ can also be expressed as follows:

$$\mathbb{E}\left(1.784^{R_n}\right) = \frac{R_n(1.784)}{n!}.$$

A lot is known about the numbers $R_{n,i}$ as well as the associated generating functions: for instance $\mathbb{E}(R_n) = \frac{2n-1}{3}$ and $\mathbb{V}(R_n) = \frac{16n-29}{90}$ (see e.g. [113]). However we cannot get our hands on $R_n(1.784)$ directly, but we can do so by exploiting a connection to the well-studied Eulerian polynomials (see e.g. [32]). The n-th Eulerian polynomial $A_n(u)$ enumerates n-permutations by their ascents and is defined as $A_n(u) = \sum_{i\geq 1} A_{n,i}u^i$, where $A_{n,i}$ is the number of n-permutations with exactly i ascents. An ascent of a permutation π is a position i for which it holds that $\pi(i) < \pi(i+1)$. Now, for the Eulerian polynomials, the following is known:

$$\sum_{n\geq 0} A_n(u)\frac{z^n}{n!} = \frac{1-u}{e^{(u-1)z} - u}. \tag{5.1.3}$$

Moreover, we have the following connection between $R_n(u)$ and $A_n(u)$ for all integers $n \geq 2$ (established in [59] and formulated more concisely by Knuth [109]):

$$\frac{R_n(u)}{n!} = \left(\frac{1+u}{2}\right)^{n-1} (1+w)^{n+1} A_n\left(\frac{1-w}{1+w}\right),$$

where $w = \sqrt{(1-u)/(1+u)}$. In order to evaluate $R_n(u)$ at $u = 1.784$, we thus only need to determine $A_n(u)$ at the corresponding value. As demonstrated in Example IX.12 in [78], it is easy to get asymptotics for the coefficients of z^n in $\sum_{n\geq 0} A_n(u)\frac{z^n}{n!}$ by a straight-forward analysis of the singularities. Indeed, for $|u| < 2$, one has:

$$\frac{A_n(u)}{n!} = \left(\frac{u-1}{\log(u)}\right)^{n+1} + \mathcal{O}(2^{-n}). \tag{5.1.4}$$

Putting Equations (5.1.3) and (5.1.4) together, we finally obtain:

$$\mathbb{E}\left(1.784^{R_n}\right) = \frac{R_n(1.784)}{n!} = \mathcal{O}\left(\left(\frac{2.784}{2} \cdot (1+w) \cdot \frac{\frac{1-w}{1+w} - 1}{\log(\frac{1-w}{1+w})}\right)^n\right) = \mathcal{O}\left(c^n\right),$$

where $w = w = \sqrt{(1-1.784)/(1+1.784)}$. Computations using any computer algebra system show that the constant $c < 1.515$. Finally, we remark that the tempting approach $\mathbb{E}\left(1.784^{R_n}\right) = 1.784^{\mathbb{E}(R_n)}$ is not correct. □

Corollary 5.15. *The runtime of the alternating run algorithm can be expected to be in*

$$\mathcal{O}\left(1.514^{\mathsf{run}(T)} \cdot n \cdot k\right).$$

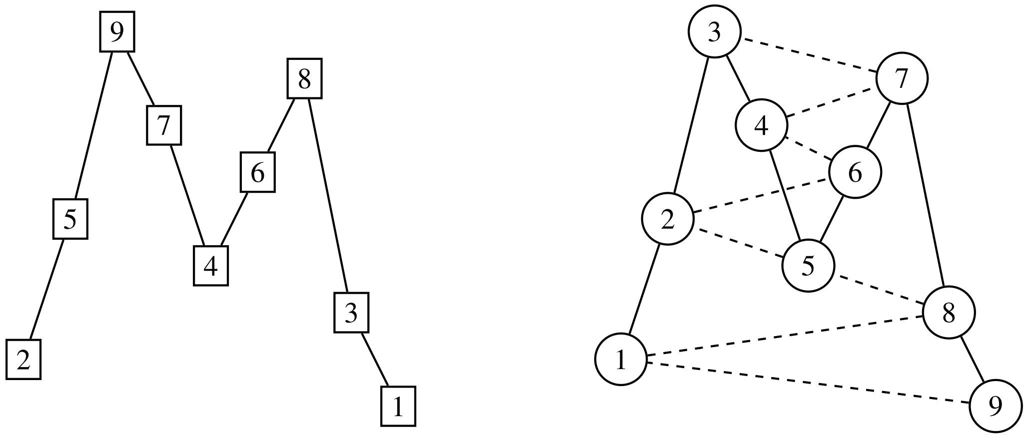


Figure 5.6: To the left is a graphical representation of the permutation π introduced in Example 5.10, to the right is the corresponding incidence graph G_π.

5.2 The Parameter $\mathsf{run}(P)$

The aim of this section is twofold: First, we want to show that PPM can be solved in time $\mathcal{O}(n^{1+\mathsf{run}(P)})$. This result builds upon an algorithm by Ahal and Rabinovich [1] and a novel connection between the pathwidth of the *incidence graph of a permutation* [1] and the number of alternating runs in that permutation. Second, we show that this runtime cannot be improved to an fpt result unless $\mathsf{FPT} = \mathsf{W[1]}$. Let us start by defining incidence graphs:

Definition 5.8. *Given an m-permutation π, the incidence graph $G_\pi = (V, E)$ of π is defined as follows: The vertices $V := [m]$ represent positions in π. There are edges between adjacent positions, i.e., $E_1 := \big\{\{i, i+1\} \mid i \in [m-1]\big\}$. There are also edges between positions where the corresponding values have a difference of* 1*, i.e., $E_2 := \big\{\{i, j\} \mid \pi(i) - \pi(j) = 1\big\}$. The edge set is defined as $E := E_1 \cup E_2$.*

Example 5.10. Consider the permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 5 & 9 & 7 & 4 & 6 & 8 & 3 & 1 \end{pmatrix}$$

written in two-line representation. A graphical representation of π can be found on the left-hand side of Figure 5.6. The corresponding graph G_π is represented on the right-hand side of the same figure. The solid lines correspond to the edges in E_1 and the dashed lines to the ones in E_2. $\dashv$

Definition 5.9. *Let $G = (V, E)$ be a simple graph, i.e., E is a set of cardinality* 2 *subsets of V. A path decomposition of G is a sequence $S_1, \ldots, S_k$ of subsets of V such that*

1. *Every vertex appears in at least one S_i, $i \in [k]$.*
2. *Every edge is a subset of at least one S_i, $i \in [k]$.*

3. *Let three indices $1 \leq h < i < j \leq k$ be given. If a vertex is contained both in S_h and S_j then it is also contained in S_i.*

The width of a path decomposition is defined as $\max\{|S_1|, \ldots, |S_k|\} - 1$. *The pathwidth of a graph G, written* $\mathsf{pw}(G)$*, is the minimum width of any path decomposition.*

In [1], Theorem 2.7 and Proposition 3.5, the authors present an algorithm that solves PPM in time $\mathcal{O}\left(n^{1+\mathsf{pw}(G_P)}\right)$. The following lemma relates $\mathsf{pw}(G_P)$ and the number of alternating runs in P.

Lemma 5.16. *For all permutations π, it holds that* $\mathsf{pw}(G_\pi) \leq \mathsf{run}(\pi)$.

Proof. Given an m-permutation π we will define a sequence $S_1, \ldots, S_m$. We then show that this sequence is a path decomposition of $G_\pi = (V, E)$ with width at most $\mathsf{run}(\pi)$. In this proof we use the variables i, j for positions in π and the variables u, v, w for values of π, i.e., $\pi(1), \pi(2)$, etc.

In order to define the sequence $S_1, \ldots, S_m$ of subsets of V, we shall extend alternating runs to maximal monotone subsequences. This means that we add the preceding valley to a run up and the preceding peak to a run down. For any $s \in [\mathsf{run}(\pi)]$, R_s then denotes the set of elements in the s-th run in π together with the preceding valley or peak. Note that this implies that $|R_s \cap R_{s+1}| = 1$ for all $s \in [\mathsf{run}(\pi) - 1]$.

We define $S'_1 := \{1\}$ and for every $v \in [2, m]$,

$$S'_v := \big\{ \max(R_j \cap [v-1]) \mid j \in [\mathsf{run}(\pi)] \text{ and } R_j \cap [v-1] \neq \emptyset \big\} \cup \big\{v\big\},$$

i.e., S'_v contains v and the largest element of every run that is smaller than v. Since S_v should contain positions in π (and not elements), we define

$$S_v := \{\pi^{-1}(w) \mid w \in S'_v\}.$$

For an example of this construction, see Example 5.11. We now check that $S_1, \ldots, S_m$ indeed is a path decomposition.

1. The vertex i appears in $S_{\pi(i)}$.

2. First we consider edges of the form $\{i, i+1\}$. Without loss of generality let $\pi(i) < \pi(i+1)$. Then $\{i, i+1\}$ is a subset of $S_{\pi(i+1)}$. Clearly, $i+1 \in S_{\pi(i+1)}$. Since $\pi(i)$ and $\pi(i+1)$ are adjacent in π there has to be an $s \in [\mathsf{run}(\pi)]$ such that $\{\pi(i), \pi(i+1)\} \subseteq R_s$. It then holds that $\max(R_s \cap [\pi(i+1)-1]) = \pi(i)$ since $\pi(i) \in R_s \cap [\pi(i+1)-1]$ and $\pi(i)$ is the largest element in R_s smaller than $\pi(i+1)$. Consequently $i \in S_{\pi(i+1)}$.

   Second, every edge $\{i, j\} \in E$ with $\pi(i) - \pi(j) = 1$ is a subset of $S_{\pi(i)}$: As before $i \in S_{\pi(i)}$. Let s be any element of $[\mathsf{run}(\pi)]$ such that $j \in R_s$. Then $\max(R_s \cap [\pi(i)-1]) = \max(R_s \cap [\pi(j)]) = \pi(j)$ and hence $j \in S_{\pi(i)}$.

   Only these two types of edges exists.

i	$\pi(i)$	X'_i	X_i
1	2	1	9
2	5	12	91
3	9	123	918
4	7	234	185
5	4	2345	1852
6	6	3456	8526
7	8	34567	85264
8	3	3 5678	8 2647
9	1	5 789	2 473

Figure 5.7: The sets $S'_1, \dots, S'_9$ and $S_1, \dots, S_9$ for the permutation $\pi = 2\,5\,9\,7\,4\,6\,8\,3\,1$

3. Let $1 \leq u < v < w \leq m$ with $i \in S_u$ and $i \in S_w$. Let s be any element of $[\mathsf{run}(\pi)]$ such that $\pi(i) \in R_s$. Then either $\pi(i) \in R_s \cap [u-1]$ or $\pi(i) = u$. In both cases is $\pi(i) \in R_s \cap [v]$. Furthermore, since $\pi(i) < w$, $\pi(i) = \max(R_s \cap [w-1]) = \max(R_s \cap [v])$. Hence $\pi(i) \in S'_v$ and $i \in S_v$.

The cardinality of each S_i is at most $\mathsf{run}(\pi) + 1$ and hence $\mathsf{pw}(G_\pi) \leq \mathsf{run}(\pi)$. □

Remark 5.17. This bound is tight since G_π for $\pi = 1\,2\,3\,\dots\,m$ is a path and hence $\mathsf{pw}(G_\pi) = \mathsf{run}(\pi) = 1$.

Example 5.11. Consider again π as defined in Example 5.10. The elements of the sets $S'_1,\dots,S'_9$ and those of $S_1, \dots, S_9$ as defined in the proof of Lemma 5.16 are given in Figure 5.11. It is easy to check that $S_1, \dots, S_9$ indeed is a path decomposition of width $4 = \mathsf{run}(\pi)$. Note that in the given table, columns of equal numbers do not contain any gaps. This fact corresponds to the third condition in the definition of path decompositions. ⊣

Theorem 5.18. PPM *can be solved in time* $\mathcal{O}(n^{1+run(P)})$.

Proof. Since $\mathsf{pw}(G_\pi) \leq \mathsf{run}(\pi)$ (Lemma 5.16), the runtime of the $\mathcal{O}(n^{1+pw(G_P)})$ algorithm can be bounded by $\mathcal{O}(n^{1+run(P)})$. □

We continue with a corresponding hardness result. We prove that one cannot hope to substantially improve the XP results in Theorem 5.18: an fpt algorithm with respect to $\mathsf{run}(P)$ is only possible if FPT = W[1].

Theorem 5.19. PPM *is* W[1]*-hard with respect to the parameter* $\mathsf{run}(P)$.

Proof. We give an fpt-reduction from the W[1]-hard SEGREGATED PERMUTATION PATTERN MATCHING problem (cf. Theorem 4.7) to PPM. We repeat the definition of SEGREGATED PERMUTATION PATTERN MATCHING here:

SEGREGATED PERMUTATION PATTERN MATCHING (SPPM)	
Instance:	An n-permutation T (the text), a k-permutation P (the pattern) and two positive integers $p \in [k]$, $t \in [n]$.
Parameter:	k
Question:	Is there a matching M of P into T such that $M(i) \leq t$ if and only if $i \leq p$?

In this problem we are looking for matchings M where for all $i \leq p$ it holds that $M(i) \in [t]$ and for all $i > p$ it holds that $M(i) \in [t+1, n]$. Let (P, T, p, t) be a SPPM instance, where $|P| = k \leq n = |T|$. We are going to construct a PPM instance $(\tilde{P}, \tilde{T})$ as follows:

$$\tilde{P} = (p + 0.5) \quad \underbrace{(k+1)(k+2)\ldots(k+n+1)}_{=R_P} \quad P$$

$$\tilde{T} = (t + 0.5) \quad \underbrace{(n+1)(n+2)\ldots\ldots(2n+1)}_{=R_T} \quad T$$

Note that the increasing runs R_P and R_T both consist of $(n+1)$ elements. The element placed at the beginning of $\tilde{P}$, $p+0.5$, is larger than p but smaller than $p+1$. Analogously, $t+0.5$ in $\tilde{T}$ is larger than t but smaller than $t+1$. Note that $\tilde{P}$ and $\tilde{T}$ are not permutations in the classical sense, since they contain elements that are not integers. However, in order to obtain permutations on $[k+n+2]$ and $[2n+2]$, we simply need to relabel the respective elements order-isomorphically.

Given this construction of $\tilde{P}$ and $\tilde{T}$ the following holds: In every matching of $\tilde{P}$ into $\tilde{T}$ the element $p + 0.5$ has to be mapped to $t + 0.5$. Indeed, the increasing run of elements $R_P = (k+1)(k+2)\ldots(k+n+1)$ in $\tilde{P}$ has to be mapped to the increasing run of elements $R_T = (n+1)(n+2)\ldots\ldots(2n+1)$ in $\tilde{T}$ and consequently P has to be matched into T. This holds because of the following observation: If the element $(k+1)$ in $\tilde{P}$ is mapped to an element $(n+i)$ with $i > 1$ in $\tilde{T}$, some of the elements of R_P have to be matched into T since R_P and R_T have the same length. This is however not possible, since all elements in T are smaller than $(n+i)$. If $(k+1)$ is instead mapped to one of the elements of T, then all remaining elements of R_P also have to be matched into T which is not possible since R_P is longer than T. Therefore, the element $(k+1)$ in $\tilde{P}$ is always mapped to the element $(n+1)$ in $\tilde{T}$. Both in $\tilde{P}$ and in $\tilde{T}$ there is only one element lying to the left of $(k+1)$ and one to left of $(n+1)$: $(p+0.5)$ and $(t+0.5)$, respectively. Thus, $(p+0.5)$ has to be mapped to $(t+0.5)$. This implies that all elements smaller than $(p+0.5)$, i.e., elements in the interval $[p]$, in P have to be mapped to elements smaller than $t+0.5$, i.e., elements in the interval $[t]$, in T. We have shown that (P, T, p, t) is a YES-instance of SPPM if and only if $(\tilde{P}, \tilde{T})$ is a YES-instance of PPM.

It remains to show that this reduction can be done in fpt-time. Clearly $\mathsf{run}(\tilde{P}) = 2 + \mathsf{run}(P) = \mathcal{O}(k)$. Moreover the length of T is bounded by a polynomial in the size of G since $|T| = n + 2 + |T| = 2n + 2 = \mathcal{O}(n)$. □

5.3 Summary

The results in this chapter are the following:

- Our main result is a fixed-parameter algorithm for PPM with a runtime of $\mathcal{O}(1.79^{\mathsf{run}(T)} \cdot n \cdot k)$. Since the combinatorial explosion is confined to $\mathsf{run}(T)$, this algorithm performs especially well when T has few alternating runs.

- Since $\mathsf{run}(T) \leq n$, this algorithm also solves PPM in time $\mathcal{O}(1.79^n \cdot n \cdot k)$. This is a major improvement over the brute-force algorithm with a runtime of $\mathcal{O}(2^n \cdot n)$.

- Since the number of runs in a random permutation is unlikely to be n, one can expect an even smaller constant than 1.79 on average. Indeed, we prove that the expected runtime of our algorithm is in $\mathcal{O}(1.52^n \cdot n \cdot k)$.

- We also show that an algorithm by Ahal and Rabinovich [1] has a runtime of $\mathcal{O}(n^{1+\mathsf{run}(P)})$. This is achieved by proving that the pathwidth of a certain graph generated by a permutation is bounded by the number of alternating runs of this permutation.

- Finally, we prove that – under standard complexity theoretic assumptions – no fixed-parameter algorithm exists with respect to $\mathsf{run}(P)$, i.e., no algorithm with a runtime of $\mathcal{O}(c^{\mathsf{run}(P)} \cdot poly(n))$ for some constant c may be hoped for. Thus, the runtime of the aforementioned $\mathcal{O}(n^{1+\mathsf{run}(P)})$ algorithm cannot be substantially improved.

Part II

Structure in Preferences

CHAPTER 6

Nearly Structured Preferences: Complexity Results

This chapter is based on the publication *Computational aspects of nearly single-peaked electorates* [70], a joint work with Gabór Erdélyi and Andreas Pfandler. We introduce and study notions of distance to single-peakedness, motivated by the fact that both experimental [116, 135] and theoretical analyses (cf. Chapter 9) have shown that single-peakedness is a property very unlikely to appear in preferences. Thus, it is reasonable to ask if preferences are close to single-peakedness.

We present a systematic theoretical study of "nearly" single-peaked preferences. Our main contributions are:

- We introduce three new notions of nearly single-peakedness. In addition, we study four notions that already have been defined or suggested in the literature.
- We explore connections between both existing and new notions by providing inequalities. These allow use to compare these notions and better understand their relationship.
- We analyze the computational complexity of computing the distance of arbitrary preference profiles to single-peakedness. In most cases we show NP-completeness. For the k-candidate deletion distance, we present a polynomial-time algorithm.
- In addition, we consider the complexity of computing distances if the axis is already given. In this case, we find polynomial-time algorithms in all considered cases.

6.1 Nearly Single-peaked Preferences

In real-world settings one can expect a certain amount of "noise" in preference data. The single-peakedness property is very fragile and thus susceptible to such noise. The following example illustrates the *fragility of single-peakedness*: Consider the single-peaked election consisting of two kinds of votes: $a \succ b \succ c \succ d$ and $d \succ c \succ b \succ a$. Assume that both votes have been cast by a large number of voters. This election is single-peaked only with respect to the axis $a > b > c > d$ and its reverse. Adding a single vote $a \succ b \succ d \succ c$ destroys the single-peakedness property although this vote is almost identical to the first kind of votes.

In this section we formally define different notions of nearly single-peakedness. All these notions define a distance measure[1] to single-peaked profiles. We will now describe them and provide first (trivial) upper bounds on these distances.

k-Voter Deletion (VD)

The first formal definition of nearly single-peaked societies was given by Faliszewski, Hemaspaandra, and Hemaspaandra [75]. Consider a preference profile $\mathcal{P}$ for which most voters are single-peaked with respect to some axis A. The voters that are not single-peaked with respect to A are referred to as *mavericks* by Faliszewski, Hemaspaandra, and Hemaspaandra. The number of mavericks, i.e., the number of voters that have to be deleted, defines a natural distance measure to single-peakedness. If an axis can be found for a large subset of the voters, this is still a fundamental observation about the structure of the preference profile.

Definition 6.1 (Faliszewski, Hemaspaandra, and Hemaspaandra [75])**.** *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is k-*voter deletion single-peaked with respect to an axis A *if by removing at most k votes from $\mathcal{P}$ one can obtain a preference profile $\mathcal{P}'$ that is single-peaked with respect to A.*

*Furthermore, we say that the profile $\mathcal{P}$ is k-*voter deletion single-peaked consistent *if there exists an axis A such that $\mathcal{P}$ is k-*voter deletion single-peaked with respect to A.

Let *VD*($\mathcal{P}$) denote the smallest k such that $\mathcal{P}$ is k-voter deletion single-peaked consistent. Note that *VD*($\mathcal{P}$) $\leq n - 1$ always holds.

Example 6.1. Consider an election with $C = \{a, b, c, d, e\}$ and $\mathcal{P} = \{V_1, V_2, \ldots, V_{202}\}$. Let the vote V_1 be defined as $a \succ_1 b \succ_1 c \succ_1 e \succ_1 d$, vote V_2 as $e \succ_2 d \succ_2 c \succ_2 a \succ_2 b$, the votes V_3 to V_{102} as $a \succ b \succ c \succ d \succ e$ and the remaining votes V_{103} to V_{202} as $e \succ d \succ c \succ b \succ a$. Notice that any preference profile containing $a \succ b \succ c \succ d \succ e$ and $e \succ d \succ c \succ b \succ a$ may only be single-peaked consistent with respect to the axis $a > b > c > d > e$ and its reverse. Since V_1 and V_2 are not single-peaked with respect to this axis, $\mathcal{P}$ is not single-peaked. Deleting V_1 and V_2 obviously yields single-peaked consistency and thus we have *VD*($\mathcal{P}$) $= 2$. $\dashv$

k-Candidate Deletion (CD)

As suggested by Escoffier, Lang, and Ötztürk [71], we introduce outlier candidates. These are candidates that do not have a "correct place" on any axis and consequently have to be deleted in order to obtain a single-peaked consistent profile. Examples could be a candidate that is not well-known (e.g., a new political party) or a candidate that prioritizes other topics than most candidates and thereby is judged by the voters according to different criteria. The votes restricted

[1] We remark that we use the words "distance" and "distance measure" with their informal meaning and not in the mathematical sense of a metric.

to the remaining candidates might still have a clear and significant structure, in particular they might be single-peaked consistent.

Definition 6.2. *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is k-*candidate deletion single-peaked* with respect to an axis A if we can obtain a set $C' \subseteq C$ by removing at most k candidates from C such that the preference profile $\mathcal{P}[C']$ is single-peaked with respect to $A[C']$.*

*Furthermore, we say that the profile $\mathcal{P}$ is k-*candidate deletion single-peaked consistent *if there exists an axis A such that $\mathcal{P}$ is k-*candidate deletion single-peaked with respect to *A.*

Let *CD*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-candidate deletion single-peaked consistent. Note that *CD*$(\mathcal{P}) \leq m - 2$ always holds.

Example 6.1 (continued)**.** Consider the preference profile $\mathcal{P}$ as defined above. Observe that for $C' = \{b, c, d\}$, $\mathcal{P}[C']$ is single-peaked consistent. Deleting a single candidate does not yield single-peaked consistency and thus *CD*$(\mathcal{P}) = 2$. $\dashv$

k-Local Candidate Deletion (LCD)

Personal friendships or hatreds between voters and candidates could move candidates up or down in a vote. These personal relationships cannot be reflected in a global axis. To eliminate the influence of personal relationships to some candidates we define a local version of the previous notion. This notion can also deal with the possibility that the least favorite candidates are ranked without special consideration or even randomly.

Definition 6.3. *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is k-*local candidate deletion single-peaked with respect to an axis *A if for every vote $V \in \mathcal{P}$ there exists a set $C' \subseteq C$ with $|C'| \geq m - k$ such that $V[C']$ is single-peaked with respect to $A[C']$.*

*Furthermore, we say that the profile $\mathcal{P}$ is k-*local candidate deletion single-peaked consistent *if there exists an axis A such that $\mathcal{P}$ is k-*local candidate deletion single-peaked with respect to *A.*

Let *LCD*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-local candidate deletion single-peaked consistent. Note that *LCD*$(\mathcal{P}) \leq m - 2$ always holds.

Example 6.1 (continued)**.** Note that it is sufficient to remove a from vote V_1 and e from vote V_2 to obtain single-peaked consistency. Consequently, *LCD*$(\mathcal{P}) = 1$. $\dashv$

k-Additional Axes (AA)

Another suggestion by Escoffier, Lang, and Ötztürk [71] was to consider the minimum number of axes such that each preference relation of the profile is single-peaked with respect to at least one of these axes. This notion is particularly useful if each candidate represents opinions on several issues (as it is the case in political elections). A voter's ranking of the candidates would then depend on which issue is considered most important by the voter and consequently each issue might give rise to its own corresponding axis.

Definition 6.4. *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is* k-additional axes single-peaked with respect to axes $A_1, \ldots, A_{k+1}$ *if there is a partition $\mathcal{P}_1, \ldots, \mathcal{P}_{k+1}$ of $\mathcal{P}$ such that for all $i \in \{1, \ldots, k+1\}$, the subprofile $\mathcal{P}_i$ is single-peaked consistent with respect to A_i.*

Furthermore, we say that the profile $\mathcal{P}$ is k-additional axes single-peaked consistent *if there exist $k+1$ axes $A_1, \ldots, A_{k+1}$ such that $\mathcal{P}$ is* k-additional axes single-peaked with respect to $A_1, \ldots, A_{k+1}$.

Let *AA*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-additional axes single-peaked consistent. Note that *AA*$(\mathcal{P}) < \min\left(n, \frac{m!}{2}\right)$ always holds. This is because the number of distinct votes is trivially bounded by n. Furthermore, *AA*$(\mathcal{P})$ is bounded by $\frac{m!}{2}$ since at most $m!$ distinct votes exist, and each vote and its reverse are single-peaked with respect to the same axes.

Example 6.1 (continued)**.** We argue that one additional axis is required for single-peaked consistency. Notice that V_1 and V_2 are single-peaked consistent with respect to axis $b > a > c > e > d$. The remaining votes are consistent with respect to $a > b > c > d > e$. Thus, one additional axis is required and hence *AA*$(\mathcal{P}) = 1$. $\dashv$

k-Global Swaps (GS)

There is a second method of dealing with candidates that are "not placed correctly" according to an axis A. Instead of deleting them from either the candidate set C or from a vote, we could try to move them to the correct position. We do this by performing a sequence of swaps of consecutive candidates. We remark that the minimum number of swaps required to change one vote to another is the *Kendall tau distance* [102] of these two votes (permutations). For example, to get from vote $abcd$ to vote $adbc$, we first have to swap candidates c and d, and then we have to swap b and d. Since this changes the votes in a more subtle way, this can be considered a less obtrusive notion than k-(Local) Candidate Deletion.

Definition 6.5. *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is* k-global swaps single-peaked with respect to an axis A *if $\mathcal{P}$ can be made single-peaked with respect to A by performing at most k swaps in the profile.*

Furthermore, we say that the profile $\mathcal{P}$ is k-global swaps single-peaked consistent *if there exists an axis A such that $\mathcal{P}$ is* k-global swaps single-peaked with respect to A.

Note that these swaps can be performed wherever we want – we can have k swaps in only one vote, or one swap each in k votes. Let *GS*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-global swaps single-peaked consistent. Note that *GS*$(\mathcal{P}) \leq \binom{m}{2} \cdot n$ always holds since rearranging a total order in order to obtain any other total order requires at most $\binom{m}{2}$ swaps.

Example 6.1 (continued)**.** It is possible to make $\mathcal{P}$ single-peaked consistent by swapping d and e in vote V_1 and swapping a and b in vote V_2. This gives *GS*$(\mathcal{P}) = 2$. $\dashv$

k-Local Swaps (LS)

We can also consider a "local budget" for swaps, i.e., we allow up to k swaps per vote. This distance measure has been introduced by Faliszewski, Hemaspaandra, and Hemaspaandra [75] as Dodgson$_k$.

Definition 6.6. *Let $E = (C, \mathcal{P})$ be an election and k a positive integer. We say that the profile $\mathcal{P}$ is k-*local swaps single-peaked with respect to an axis A *if $\mathcal{P}$ can be made single-peaked with respect to A by performing no more than k swaps per vote.*

*Furthermore, we say that the profile $\mathcal{P}$ is k-*local swaps single-peaked consistent *if there exists an axis A such that $\mathcal{P}$ is k-*local swaps single-peaked with respect to A.

Let *LS*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-local swaps single-peaked consistent. Note that *LS*$(\mathcal{P}) \leq \binom{m}{2}$ always holds.

Example 6.1 (continued)**.** Since only one swap is required in V_1 and V_2 each, we consequently obtain *LS*$(\mathcal{P}) = 1$. ⊣

k-Candidate Partition (CP)

Our last nearly single-peaked notion is the candidate analogon of k-additional axes. In this case we partition the set of candidates into subsets such that all of the corresponding profiles are single-peaked consistent. This notion is useful for example in the following situation. Each candidate has an opinion on a controversial Yes/No-issue. Depending on their own preference voters will always rank all Yes-candidates before or after all No-candidates. It might be that when considering only the Yes- or only the No-candidates, the election is single-peaked. Therefore, if we acknowledge the importance of this Yes/No-issue and partition the candidates accordingly, we may obtain two single-peaked elections.

Definition 6.7. *Let $E = (C, \mathcal{P})$ be an election, k be a positive integer, and a partition $C_1, \dots, C_k$ be disjoint subsets of C such that $C_1 \cup \dots \cup C_k = C$. We say that the profile $\mathcal{P}$ is k-*candidate partition single-peaked with respect to an axis A and a partition $C_1, \dots, C_k$ *if for each $i \in \{1, \dots, k\}$ the profile $\mathcal{P}[C_i]$ is single-peaked with respect to $A[C_i]$.*

*Furthermore, we say that the profile $\mathcal{P}$ is k-*candidate partition single-peaked consistent *if there exist an axis A and a partition $C_1, \dots, C_k$ such that $\mathcal{P}$ is k-*candidate partition single-peaked with respect to A and $C_1, \dots, C_k$.

Let *CP*$(\mathcal{P})$ denote the smallest k such that $\mathcal{P}$ is k-candidate partition single-peaked consistent. Note that *CP*$(\mathcal{P}) \leq \lceil \frac{m}{2} \rceil$ always holds.

Example 6.1 (continued)**.** We partition the candidates into $C_1 = \{a, e\}$ and $C_2 = \{b, c, d\}$. Notice that $\mathcal{P}[C_1]$ is trivially single-peaked consistent because this holds for all profiles over at most two candidates. Furthermore, $\mathcal{P}[C_2]$ contains only votes of the form $b \succ c \succ d$ or its reverse, which also gives immediately single-peakedness. Thus, *CP*$(\mathcal{P}) = 2$. ⊣

Decision Problems

We now introduce the algorithmic problems we will study. For X $\in$ {Voter Deletion, Candidate Deletion, Local Candidate Deletion, Additional Axes, Global Swaps, Local Swaps, Candidate Partition} we define:

X SINGLE-PEAKED CONSISTENCY	
Instance:	An election $E = (C, \mathcal{P})$ and a positive integer k.
Question:	Is $\mathcal{P}$ k-X single-peaked consistent?

X SINGLE-PEAKED EVALUATION	
Instance:	An election $E = (C, \mathcal{P})$, a positive integer k and an axis[‡] A.
Question:	Is $\mathcal{P}$ k-X single-peaked with respect to A?

Clearly, the X SINGLE-PEAKED EVALUATION is computationally at most as hard as X SINGLE-PEAKED CONSISTENCY, since the evaluation problem has the axis as additional input.

6.2 Basic Results about Single-Peaked Profiles

We start with a simple observation which we will use in several proofs.

Lemma 6.1. *Let $\mathcal{P}$ be a preference profile containing the vote $V : c_1 \dots c_m$ and its reverse $\overline{V}$. Then $\mathcal{P}$ is either single-peaked with respect to the axis $c_1 < \dots < c_m$ (and its reverse) or it is not single-peaked at all.*

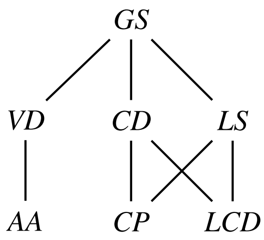
Proof. Since the vote V ranks c_m last while the vote $\overline{V}$ ranks c_1 last, these candidates have to be at the left-most and right-most position on any compatible axis. Note that c_1 is top-ranked in V. Hence this already determines the position of all other candidates. Consequently only two axes are possible: $c_1 < \dots < c_m$ and $c_m < \dots < c_1$. □

The following observation says that any subelection, i.e., an election with the same voters over a subset of the candidate set, of a single-peaked election is also single-peaked.

Lemma 6.2. *Let $(C, \mathcal{P})$ be a given election and $C' \subseteq C$. If $\mathcal{P}$ is single-peaked consistent then also $\mathcal{P}[C']$ is single-peaked consistent.*

Proof. This is an immediate consequence of the definition of single-peakedness, Definition 2.2, since valleys cannot appear by restricting $\mathcal{P}$ to C'. □

[‡]For Additional Axes we assume that $k + 1$ axes $A_1, \dots, A_{k+1}$ are given in the input (cf. Definition 6.4). For Candidate Partition we assume that an axis A together with a partition $C_1, \dots, C_k$ is given in the input (cf. Definition 6.7).



GS ... Global Swaps
VD ... Voter Deletion
CD ... Candidate Deletion
LS ... Local Swaps
AA ... Additional Axes
CP ... Candidate Partition
LCD ... Local Candidate Deletion

Figure 6.1: Hasse diagram of the partial order described in Theorem 6.3

6.3 Relations between Notions of Nearly Single-Peakedness

Theorem 6.3 shows several inequalities that hold for the distance measures under consideration. We hereby show how these measures relate to each other.

Theorem 6.3. *Let $\mathcal{P}$ be a preference profile. Then the following inequalities hold:*

(1) $LS(\mathcal{P}) \leq GS(\mathcal{P})$.
(2) $LCD(\mathcal{P}) \leq CD(\mathcal{P})$.
(3) $CD(\mathcal{P}) \leq GS(\mathcal{P})$.
(4) $LCD(\mathcal{P}) \leq LS(\mathcal{P})$.
(5) $VD(\mathcal{P}) \leq GS(\mathcal{P})$.
(6) $AA(\mathcal{P}) \leq VD(\mathcal{P})$.
(7) $CP(\mathcal{P}) \leq CD(\mathcal{P}) + 1$.
(8) $CP(\mathcal{P}) \leq LS(\mathcal{P}) + 1$.

This list is complete in the following sense: Inequalities that are not listed here and that do not follow from transitivity do not hold in general. The resulting partial order with respect to $\leq$ is displayed in Figure 6.1 as a Hasse diagram.

Proof. Inequalities 1 and 2 are immediate consequences of the definitions since k-*LS* permits more swaps than k-*GS* and k-*LCD* permits more candidate deletions than k-*CD*. Inequalities 3 and 4 are due to the fact that swapping two candidates in a vote is at most as effective as removing one of these candidates. Similarly, for Inequality 5 observe that removing the corresponding voter is at least as effective as swapping two candidates in the vote. Concerning Inequality 6 observe that instead of deleting a voter we can always add an additional axis for this voter. Inequality 7 follows from the fact that putting each deleted candidate in its own partition leads to single-peakedness if deleting these candidates does.

In order to show Inequality 8 let $\mathcal{P}$ be k-local swaps single-peaked consistent. This means that there exists an axis A such that after performing at most k swaps per voter, $\mathcal{P}$ becomes single-peaked with respect to A. Without loss of generality assume that the axis A is $c_1 < c_2 < \dots < c_m$. We now partition the candidates in $k+1$ sets $S_0, \dots, S_k$. This is done by putting the i-th smallest element of A into the (i modulo $k+1$)-th set. Since we assume that A is $c_1 < c_2 < \dots < c_m$, we can equivalently say that c_i is put into the (i modulo $k+1$)-th set, i.e., the c_1 in S_1, the c_2 in S_2, the c_k in S_k and c_{k+1} in S_0. Let $S \in \{S_0, \dots, S_k\}$. Towards a contradiction assume that $\mathcal{P}[S]$ is not single-peaked with respect to $A[S]$. By Definition 2.2 there exists some voter $V \in \mathcal{P}$ and three candidates $c_i, c_j, c_k \in C$ such that $c_i < c_j < c_k$ on axis $A[S]$ (or equivalently $i < j < k$), $c_i \succ c_j$ and $c_k \succ c_j$. On axis A the distance between c_i and

	VD ($\mathcal{P}$)	*CD* ($\mathcal{P}$)	*LCD* ($\mathcal{P}$)	*GS* ($\mathcal{P}$)	*LS* ($\mathcal{P}$)	*AA* ($\mathcal{P}$)	*CP* ($\mathcal{P}$)
Voter Deletion . . . . *VD* ($\mathcal{P}$)	=	1	4	$\leq$	4	1	1
Candidate Deletion . . . . *CD* ($\mathcal{P}$)	3	=	4	$\leq$	4	3	3
Local Candidate Deletion . . . *LCD* ($\mathcal{P}$)	3	$\leq$	=	$\leq$	$\leq$	3	3
Global Swaps . . . . *GS* ($\mathcal{P}$)	2	2	2	=	4	2	2
Local Swaps . . . . *LS* ($\mathcal{P}$)	2	2	2	$\leq$	=	2	2
Additional Axes . . . . *AA* ($\mathcal{P}$)	$\leq$	5	5	$\leq$	6	=	5
Candidate Partition . . . . *CP* ($\mathcal{P}$)	8	$\leq$	7	$\leq$	$\leq$	8	=

Table 6.1: Inequalities regarding the distance measures. This table should be read as follows. Measures on the left-most column are bounded ($\leq$) by the measures in the top row. Numbers point to the corresponding counterexamples if no such bound exists.

c_j respectively c_j and c_k is at least $k+1$, i.e., at least k elements lie in between them. We know that at most k swaps in V can make this vote single-peaked with respect to A. Let V' denote this swapped vote. Necessarily, these swaps have to either cause that $c_j \succ' c_{j-1} \succ' \cdots \succ' c_{i+1} \succ' c_i$ holds or that $c_j \succ' c_{j+1} \succ' \cdots \succ' c_{k-1} \succ' c_k$ holds in V'_v (depending whether top-ranked candidate of V'_v is right or left of c_j). Let us focus on the case where the swaps ensure that $c_j \succ' c_{j-1} \succ' \cdots \succ' c_{i+1} \succ' c_i$; the other case is analogous. For V, contrary to V', it holds that $c_i \succ c_j$. Hence these swaps have to cause that $c_j \succ' c_i$ holds. In addition, at least k elements, namely $c_{i+1}, \ldots, c_{j-1}$, have to be in between them. This requires at least $k+1$ swaps which contradicts the fact that at most k swaps suffice. Therefore for all partition sets $S \in \{S_0, \ldots, S_k\}$, $\mathcal{P}[S]$ is single-peaked consistent and $\mathit{CP}(\mathcal{P}) \leq \mathit{LS}(\mathcal{P}) + 1$.

It remains to show that these are indeed all inequalities. This can be done by providing counterexamples for each remaining case. Table 6.1 offers an overview by pointing to the corresponding counterexample. In the following examples we assume that $m, n \geq 4$.

Counterexample 1 (VD cannot be bounded by CD, AA and CP): Consider the preference profile over the candidate set $C = \{c_1, \ldots, c_m\}$ with the following $2m$ votes:

- There are m votes of the form: $c_1\ c_2\ \ldots\ c_m$.
- There are m votes of the form: $c_m\ c_2\ c_3\ \ldots\ c_{m-1}\ c_1$.

The corresponding preference profile $\mathcal{P}$ is not single-peaked consistent. This is because c_2 has to be next to both c_1 and c_m on any suitable axis but c_1 and c_m have to be either the left-most or right-most element. Consequently, $\mathit{VD}(\mathcal{P}) = m$. Removing candidates instead of voters is far more useful in this case. When we remove either c_1 or c_m, $\mathcal{P}$ becomes single-peaked and hence $\mathit{CD}(\mathcal{P}) = 1$. Since we have only two distinct votes, we require two axes to make $\mathcal{P}$ single-peaked and hence $\mathit{AA}(\mathcal{P}) = 1$. Furthermore, notice that we can obtain single-peaked consistency by partitioning the candidates into two sets $C_1 = \{c_1, c_m\}$ and $C_1 = \{c_2, \ldots, c_{m-1}\}$. As a consequence $\mathit{CP}(\mathcal{P}) = 2$.

Counterexample 2 (Neither GS nor LS can be bounded by AA, CD, LCD and CP): This counterexample is similar to the previous one but $\mathcal{P}$ consists of only two votes. Let the set of candidates be $C = \{c_1, \ldots, c_{3m+1}\}$.

- There is one vote of the form: $c_1 \; c_2 \; \ldots \; c_{3m+1}$.
- There is one vote of the form: $c_{3m+1} \; c_2 \; c_3 \; \ldots \; c_{3m} \; c_1$.

If we consider this profile $\mathcal{P}$ restricted to the candidates c_1, c_{m+1}, c_{2m+1} and c_{3m+1}, i.e., $\mathcal{P}\left[\{c_1, c_{m+1}, c_{2m+1}, c_{3m+1}\}\right]$, we observe that this restricted profile is not single-peaked. Consequently, by Lemma 6.2, $\mathcal{P}$ is not single-peaked as well. If we want to make $\mathcal{P}$ single-peaked with swaps, at least two of $\{c_1, c_{m+1}, c_{2m+1}, c_{3m+1}\}$ have to swap position. This requires at least m swaps and consequently $\mathit{GS}(\mathcal{P}) \geq \mathit{LS}(\mathcal{P}) \geq m$. Since there are only two votes, $\mathit{AA}(\mathcal{P}) = \mathit{VD}(\mathcal{P}) = 1$. As in the previous counterexample removing either c_1 or c_{3m+1} yields a single-peaked profile and hence $\mathit{CD}(\mathcal{P}) = 1$. Since $\mathit{CP}(\mathcal{P}) \leq \mathit{CD}(\mathcal{P})$ also $\mathit{CP}(\mathcal{P}) = 1$. Finally, $\mathit{LCD}(\mathcal{P}) \leq \mathit{CD}(\mathcal{P})$ implies $\mathit{LCD}(\mathcal{P}) = 1$.

Counterexample 3 (Neither CD nor LCD can be bounded by VD, AA and CP): This time we consider three votes over the candidates $C = \{c_1, \ldots, c_{2m}\}$.

- There is one vote V_1 of the form: $c_1 \; c_2 \; \ldots \; c_{2m}$.
- There is one vote V_2 of the form: $c_{2m} \; c_{2m-1} \; \ldots \; c_1$.
- There is one vote V_3 of the form: $c_m \; \ldots \; c_1 \; c_{m+1} \; \ldots \; c_{2m}$.

By Lemma 6.1 we only have to consider the axis $c_1 < c_2 < \cdots < c_{2m}$ for $\mathcal{P} = (V_1, V_2, V_3)$. The third vote V_3 is however not single-peaked with respect to this axis. Hence $\mathcal{P}$ is not single-peaked consistent. Here $\mathit{VD}(\mathcal{P}) = 1$ since deleting vote V_3 leads to single-peaked consistency. Since $\mathit{AA}(\mathcal{P}) \leq \mathit{VD}(\mathcal{P})$ also $\mathit{AA}(\mathcal{P}) = 1$. However, we need to remove by far more candidates; we have to remove candidates until the indices of the remaining candidates in V_3 are either increasing or decreasing. Thus, there are at least $m - 1$ to remove and $\mathit{CD}(\mathcal{P}) \geq \mathit{LCD}(\mathcal{P}) \geq m - 1$. Finally, we have that $\mathit{CP}\,(\mathcal{P}) = 2$ since $\mathcal{P}[\{c_1, \ldots, c_m\}]$ and $\mathcal{P}[\{c_{m+1}, \ldots, c_{2m}\}]$ are single-peaked consistent.

Counterexample 4 (Neither VD, GS nor CD can be bounded by LCD and LS): We consider an election with $3n$ votes over the candidates $C = \{c_1, \ldots, c_{3n}\}$.

- There are n votes $V_1, \ldots, V_n$ of the form: $c_1 \; c_2 \; \ldots \; c_{3n}$.
- There are n votes $V_{n+1}, \ldots, V_{2n}$ of the form: $c_{3n} \; c_{3n-1} \; \ldots \; c_1$.
- The remaining votes are obtained from the first vote by swapping the first two candidates in each block consisting of three candidates. Formally, for each $i \in \{1, \ldots, n\}$ there is a vote V_{2n+i} of the form: $c_1 \; \ldots \; c_{3(i-1)} \; c_{3(i-1)+2} \; c_{3(i-1)+1} \; c_{3i} \; \ldots \; c_{3n}$.

Let $\mathcal{P} = (V_1, V_2, \ldots, V_{3n})$. By using Lemma 6.1 it is easy to check that for each $1 \leq i \leq n$, $\mathcal{P}[\{c_{3(i-1)+2}, c_{3(i-1)+1}, c_{3i}\}]$ is not single-peaked consistent. By Lemma 6.2, $\mathcal{P}$ is not single-peaked consistent. Also, this implies that we have to remove at least one candidate in each set $\{c_{3(i-1)+2}, c_{3(i-1)+1}, c_{3i}\}$ in order to make $\mathcal{P}$ single-peaked consistent. Therefore $\mathit{CD}(\mathcal{P}) \geq n$. Since $\mathit{GS}(\mathcal{P}) \geq \mathit{CD}(\mathcal{P})$ also $\mathit{GS}(\mathcal{P}) \geq n$. We now want to prove a lower bound on $\mathit{VD}(\mathcal{P})$. If we delete $n-1$ votes then at least one vote of $\{V_1, \ldots, V_n\}$, one of $\{V_{n+1}, \ldots, V_{2n}\}$ and one of $\{V_{2n+1}, \ldots, V_{3n}\}$ remains. Again by Lemma 6.2, this profile would not be single-peaked consistent. Hence $\mathit{VD}(\mathcal{P}) > n-1$. Finally, notice that the votes $V_{2n+1}, \ldots, V_{3n}$ can be turned into vote V_1 by a single swap, which shows that $\mathit{LS}(\mathcal{P}) = 1$. Since $\mathit{LCD}(\mathcal{P}) \leq \mathit{LS}(\mathcal{P})$ also $\mathit{LCD}(\mathcal{P}) = 1$.

Counterexample 5 (AA cannot be bounded by CD, LCD and CP): In this example we use n votes over the candidates $C = \{c_1, \ldots, c_{n+1}\}$, where $n \geq 3$.

- For each $i \in \{1, \ldots, n\}$, there is one vote V_i of the form:
  $c_{n+1}\ c_i\ c_{i-1}\ \ldots\ c_1\ c_{i+1}\ c_{i+2}\ \ldots\ c_n.$

Let us consider the preference profile $\mathcal{P} = (V_1, V_2, \ldots, V_n)$. All votes have the same peak but different candidates in the second position. If this preference profile was single-peaked then these second-place candidates had to be either left or right of the peak. This is not possible for three or more candidates. Hence the profile $\mathcal{P}$ containing three or more votes is not single-peaked. By the previous argument $\mathit{AA}(\mathcal{P}) \geq \frac{n}{3}$. Deleting c_{n+1} however makes $\mathcal{P}$ single-peaked with respect to the axis $c_1 < c_2 < \cdots < c_n$ and hence $\mathit{CD}(\mathcal{P}) = \mathit{LCD}(\mathcal{P}) = \mathit{CP}(\mathcal{P}) = 1$.

Counterexample 6 (AA cannot be bounded by LS): We consider n votes over $4n$ candidates $C = \{c_1, \ldots, c_{4n}\}$.

- For each $i \in \{1, \ldots, n\}$, there is one vote V_i of the form:
  $c_1\ \ldots\ c_{4i-4}\ c_{4i}\ c_{4i-2}\ c_{4i-1}\ c_{4i-3}\ c_{4i+1}\ \ldots\ c_{4n}.$

Let $\mathcal{P} = (V_1, \ldots, V_n)$. The preference profile $\mathcal{P}$ is not single-peaked consistent since $\mathcal{P}[\{c_{4k-3}, c_{4k-2}, c_{4k-1}, c_{4k}\}]$ is neither for any $k \in \{1, \ldots, n\}$. With 5 swaps in each vote we can make these votes identical and hence $\mathit{LS}(\mathcal{P}) \leq 5$. Also, a profile consisting of only two of these votes is not single-peaked; hence $\mathit{AA}(\mathcal{P}) \geq \frac{n}{2}$.

Counterexample 7 (CP cannot be bounded by LCD): Consider an election with $3n$ votes over the candidates $C = \{c_1, \ldots, c_{3n}\}$.

- For each $i \in \{1, \ldots, 3n\}$, there is one vote V_i of the form: $c_1\ \ldots\ c_{i-1}\ c_{i+1}\ \ldots\ c_{3n}\ c_i.$

Since the lowest ranked candidates have to be either at the left-most or right-most position on the axis and there are more than two lowest ranked candidates, this profile is not single-peaked consistent. However, if the last ranked-candidate is removed in each vote, the profile becomes single-peaked consistent and hence $\mathit{LCD}(\mathcal{P}) = 1$. Concerning $\mathit{CP}(\mathcal{P})$ notice that any partition into n sets contains a set with at least three candidates – say c_i, c_j and c_k. But then the votes V_i, V_j and V_k cannot be single-peaked consistent because they rank three different candidates at the last position. Hence n candidate partitions are not enough to obtain single-peaked consistency and hence *CP*$(\mathcal{P}) > n$.

Counterexample 8 (CP cannot be bounded by VD and AA): Consider the candidates $C = \{c_1, \ldots, c_{m^2}\}$ and the following three votes:

- There is one vote V_1 of the form: $c_1 \ c_2 \ \ldots \ c_{m^2}$.
- There is one vote V_2 of the form: $c_{m^2} \ c_{m^2-1} \ \ldots \ c_1$.
- There is one vote V_3 of the form:
  $c_1 \ c_{m+1} \ c_{2m+1} \ \cdots \ c_{m(m-1)+1} \ c_2 \ c_{m+2} \ c_{2m+2} \ \cdots \ c_{m(m-1)+2} \ \cdots$
  $\cdots \ c_m \ c_{2m} \ c_{3m} \ \cdots \ c_{m^2}$.

This preference profile is not single-peaked but *VD* $(\mathcal{P}) = 1$ and *AA* $(\mathcal{P}) = 1$. The candidates, however, have to be partitioned into many sets in order to obtain single-peakedness. First, observe that by Lemma 6.1 we only have to consider the axis $c_1 > c_2 > \cdots > c_{m^2}$. Let us now consider vote V_3. Since we have fixed an axis we can consider longest increasing and decreasing subsequences in this vote. Note that both increasing and decreasing subsequences have a length of less than $2m$. Hence a subset of the candidates cannot be single-peaked if it contains more than $4m$ candidates. We therefore have to partition the candidates of $\mathcal{P}$ into sets of cardinality at most $4m$ and by that *CP* $(\mathcal{P}) \geq \frac{m}{4}$. $\square$

We conclude this section by illustrating how the inequalities stated in Theorem 6.3 can be used to obtain new results. More specifically, we will show that it is possible to construct preference profiles such that the profile is close to single-peakedness but does not have a Condorcet winner.

Theorem 6.4. *For every $m \geq 3$ and $n \geq 1$ there is an election $E = (C, \mathcal{P})$ with $2n + 1$ votes and m candidates such that*

- *GS*$(\mathcal{P}) = 1$ *and*
- *$\mathcal{P}$ does not have a Condorcet winner.*

Proof. Let the set of candidates be $C = \{a, b, c\} \cup \{d_1, \ldots, d_{m-3}\}$, where $d_1, \ldots, d_{m-3}$ are dummy candidates. The profile $\mathcal{P}$ contains the following votes:

- a single vote of the form: $b \succ c \succ a \succ d_1 \succ \cdots \succ d_{m-3}$,
- n votes of the form: $a \succ b \succ c \succ d_1 \succ \cdots \succ d_{m-3}$, and
- n votes of the form: $c \succ a \succ b \succ d_1 \succ \cdots \succ d_{m-3}$.

It is straight-forward to verify that the profile $\mathcal{P}$ does not have a Condorcet winner. Notice that $\mathcal{P}$ becomes single-peaked with respect to axis $b > a > c > d_1 > \cdots > d_{m-3}$ if we swap candidates a and c in the single vote. Hence, we know that *GS*$(\mathcal{P}) = 1$. $\square$

Due to the inequalities stated in Theorem 6.3, the result of Theorem 6.4 holds also if *GS*$(\mathcal{P})$ is replaced by any of the measures *VD*, *CD*, *LCD*, *LS*, *AA*, and *CP*. Therefore, even a distance of 1 to single-peakedness (with respect to the measures discussed in this work) does not help to avoid the Condorcet-paradox.

6.4 Complexity of Nearly Single-Peaked Consistency

6.4.1 Hardness results

We start with the complexity analysis of voter deletion single-peaked consistency. In our proof we are going to cascade two or more preference profiles. The following definition captures this notion.

Definition 6.8. *Let $(C_1, \mathcal{P}_1)$ and $(C_2, \mathcal{P}_2)$ be two elections with $C_1 \cap C_2 = \emptyset$. Furthermore, let $\mathcal{P}_1 = (V_1', \ldots, V_n')$ and $\mathcal{P}_2 = (V_1'', \ldots, V_n'')$. We define $\mathcal{P}_1 \mathbin{\bigcirc\hspace{-0.8em}\succ} \mathcal{P}_2 = (V_1, \ldots, V_n)$, where for any $i \in \{1, \ldots, n\}$ the total order V_i is defined by*

$$c \succ_i c' \text{ iff } (c, c' \in C_1 \text{ and } c \succ_i' c') \text{ or } (c, c' \in C_2 \text{ and } c \succ_i'' c') \text{ or } (c \in C_1 \text{ and } c' \in C_2).$$

Note that $\mathcal{P}_1 \mathbin{\bigcirc\hspace{-0.8em}\succ} \mathcal{P}_2$ is always a preference profile over $C_1 \cup C_2$.

Lemma 6.5. *Let $(C_1, \mathcal{P}_1)$ and $(C_2, \mathcal{P}_2)$ be two elections with $C_1 \cap C_2 = \emptyset$. Assume that*

- *$\mathcal{P}_1$ and $\mathcal{P}_2$ are single-peaked consistent with respect to the axes A_1 and A_2, respectively.*
- *The votes in $\mathcal{P}_2$ have at most 2 peaks.*
- *These (two) peaks are adjacent on the axis A_2.*

Then $\mathcal{P}_1 \mathbin{\bigcirc\hspace{-0.8em}\succ} \mathcal{P}_2$ is single-peaked.

Proof. We are going to construct an axis A in a way that $\mathcal{P}_1 \mathbin{\bigcirc\hspace{-0.8em}\succ} \mathcal{P}_2$ is single-peaked with respect to A. First we split A_2 in two parts A_2' and A_2''. If $\mathcal{P}_2$ contains two peaks (which have to be adjacent), we split A_2 in between these two peaks. If $\mathcal{P}_2$ contains only one peak, we split A_2 left of the peak (this is arbitrary). The new axis A is A_2' followed by A_1 and then A_2'', i.e., $A_2' > A_1 > A_2''$. The correctness proof of this construction is straight-forward. □

Before we start with the hardness proof, let us first make the following observation.

Observation 6.6. We are given a set of candidates $C = \{a, b, c, d\}$ and three votes V_v, V_e and V_{ne}, where the candidates are ranked as follows:

- $a \succ_v c \succ_v b \succ_v d$,
- $c \succ_e b \succ_e d \succ_e a$ and
- $d \succ_{ne} c \succ_{ne} b \succ_{ne} a$.

Then the preference profile (V_v, V_e) is single-peaked with respect to the axis $a > c > b > d$ and (V_e, V_{ne}) is single-peaked with respect to the axis $d > c > b > a$. The profile (V_v, V_{ne}) is not single-peaked consistent.

We show NP-hardness via a reduction from the clique problem, one of the standard NP-complete problems. This result has been proven independently by Bredereck [43] and was subsequently published in a more general context [44].

CLIQUE	
Instance:	A graph (V_G, E_G) and a positive integer s.
Question:	Has the (V_G, E_G) an induced subgraph of size s that is complete, i.e., does (V_G, E_G) contain a clique of size s?

Theorem 6.7. VOTER DELETION SINGLE-PEAKED CONSISTENCY *is NP-complete.*

Proof. To show hardness we reduce from CLIQUE. Let $G = (V_G, E_G)$ be the graph in which we look for a clique of size s. Furthermore, let $V_G = \{v_1, \ldots, v_n\}$ be the set of vertices and E_G the set of edges. Each vertex v_i has four corresponding candidates $c_i^1, \ldots, c_i^4$. We consequently have $C = \{c_1^1, \ldots, c_1^4, c_2^1, \ldots, c_2^4, \ldots, c_n^1, \ldots, c_n^4\}$. The votes directly correspond to vertices and thus $\mathcal{P} = (V_1, \ldots, V_n)$.

In order to define the votes in $\mathcal{P}$ we introduce three functions creating partial votes. In the following definition let $a, b, c, d \in C$.

$$\begin{aligned} f_v(a,b,c,d) &= a \succ c \succ b \succ d \\ f_e(a,b,c,d) &= c \succ b \succ d \succ a \\ f_{ne}(a,b,c,d) &= d \succ c \succ b \succ a \end{aligned}$$

If we consider f_v, f_e and f_{ne} as votes then observe that by Observation 6.6 (f_v, f_e) and (f_e, f_{ne}) are single-peaked consistent but (f_v, f_{ne}) is not.

Next we define a mapping $p(i,j)$ to a total order over the candidates $\{c_j^1, \ldots, c_j^4\}$.

$$p(i,j) = \begin{cases} f_v(c_j^1, c_j^2, c_j^3, c_j^4) & \text{if } i = j \\ f_e(c_j^1, c_j^2, c_j^3, c_j^4) & \text{if } \{i,j\} \in E_G \\ f_{ne}(c_j^1, c_j^2, c_j^3, c_j^4) & \text{if } \{i,j\} \notin E_G \end{cases}$$

The intuition behind function $p(i,j)$ is to encode a row of the adjacency matrix of G as a vote in the preference profile $\mathcal{P}$. To this end, we put in "cell" (i,j) the result of f_e if there is an edge between i and j. In case there is no edge between i and j we put the result of f_{ne} in cell (i,j). In the special case $i = j$ (we are in the diagonal of the matrix) we put the result of f_v in the cell.

Let the partial profiles representing the columns of the adjacency matrix be defined as $\mathcal{P}_j = (p(1,j), \ldots, p(n,j))$, for $j \in \{1, \ldots, n\}$. We are now going to define the preference profile $\mathcal{P} = (V_1, \ldots, V_n)$ by

$$\mathcal{P} = \mathcal{P}_1 \circledsucc \mathcal{P}_2 \circledsucc \cdots \circledsucc \mathcal{P}_n.$$

To conclude the construction let $E = (C, \mathcal{P})$ and $k = n - s$, i.e., we are allowed to delete k voters from E in order to obtain a single-peaked profile. The intuition behind the construction is that the voters in a single-peaked profile will correspond to a clique. We claim that G has a clique of cardinality s if and only if it is possible to remove k voters from $\mathcal{P}$ in order to make the resulting preference profile single-peaked consistent.

"$\Rightarrow$" Assume that there is a clique $I = \{v_{i_1}, \ldots, v_{i_s}\}$ with $|I| = s$. Let $\mathcal{P}' = (V_{i_1}, \ldots, V_{i_s})$. By that we keep only those voters whose corresponding vertices are contained in the clique I. Observe that the election $E' = (C, \mathcal{P}')$ can be obtained by deleting $k = n - s$ voters from the election E, $|V \setminus I| = k$. It remains to show that E' is indeed single-peaked consistent. Recall that we denoted the votes in the j-th "column" of the profile by $\mathcal{P}_j$. By $\mathcal{P}'_j$ we denote the j-th "column" of a profile considering only the voters from $\mathcal{P}'$. Since I is a clique, for each $x, y \in I$, $x \neq y$, there is an edge $\{x, y\} \in E_G$. Thus the profile cannot contain an instantiation of f_v and of f_{ne} in the same column. By Observation 6.6, all profiles $\mathcal{P}_j$ with $j \in \{1, \ldots, n\}$ are single-peaked consistent. In order to be able to apply Lemma 6.5, all conditions have to be checked. First, notice that the profiles $\mathcal{P}'_j$ and $\mathcal{P}'_{j'}$, for $1 \leq j < j' \leq n$, do not share any candidates and are single-peaked consistent. Furthermore, each of the profiles has at most two peaks. Each column contains either instantiations of f_v and f_e or instantiations of f_e and f_{ne}. Otherwise it would not be single-peaked consistent. But then there are only two top-ranked candidates, i.e., either the candidates top-ranked by f_v and f_e, or the candidates top-ranked by f_e and f_{ne}. Finally, the two top-ranked candidates of $\mathcal{P}'_j$ have to be adjacent on the axis which gives single-peaked consistency. Consider again Observation 6.6. For (f_v, f_e) the top-ranked candidates a and c are adjacent on the axis $a > c > b > d$. The same holds for (f_e, f_{ne}) with axis $d > c > b > a$ and c, d as top-ranked candidates. Since all conditions are fulfilled, we can iteratively apply Lemma 6.5. Therefore, $\mathcal{P}'_1 \circledcirc \mathcal{P}'_2, (\mathcal{P}'_1 \circledcirc \mathcal{P}'_2) \circledcirc \mathcal{P}'_3, \ldots, (\mathcal{P}'_1 \circledcirc \cdots) \circledcirc \mathcal{P}'_n$ and hence also $\mathcal{P}'$ are single-peaked consistent.

"$\Leftarrow$" Assume that $E' = (C, \mathcal{P}')$ is an election that has been obtained from E by deleting k voters such that $\mathcal{P}'$ is single-peaked. Consequently $\mathcal{P}'$ contains s votes. Let $i_1, \ldots, i_s \in \{1, \ldots, n\}$ such that $\mathcal{P}' = (V_{i_1}, \ldots, V_{i_s})$.

We claim that $\mathcal{P}'$ is a clique in G. By Lemma 6.2 we know that each of the n columns $(\mathcal{P}'_1, \ldots, \mathcal{P}'_n)$ of $\mathcal{P}'$ is single-peaked consistent. Then, by Observation 6.6, each column must not contain an instance of f_v together with an instance of f_{ne}. (Otherwise the respective column would not be single-peaked consistent!) Observe that by construction each vote (in $\mathcal{P}'$) contains an instance of f_v in some column. But then each vertex must be adjacent to all other vertices – in other words the vertices $v_{i_1}, \ldots, v_{i_s}$ form a clique. $\square$

We now turn to additional axes single-peaked consistency. Here we make use of a similar construction as presented in Theorem 6.7 with the difference that we now show NP-hardness via a reduction from the partition into cliques problem, which is also one of the standard NP-complete problems (see, e.g., [86]).

PARTITION INTO CLIQUES	
Instance:	A graph (V_G, E_G) and a positive integer s.
Question:	Is it possible to partition V_G into s sets such that each set of vertices induces a clique on (V_G, E_G)?

Theorem 6.8. ADDITIONAL AXES SINGLE-PEAKED CONSISTENCY *is* NP*-complete.*

Proof. Hardness is shown by a reduction from PARTITION INTO CLIQUES. For the reduction we use the same transformation as presented in the proof of Theorem 6.7 to obtain an election. Then we set $k = s - 1$, i.e., we are searching for a partition of the voters into s disjoint sets such that each of the partitions is single-peaked consistent. Due to the one-to-one correspondence between voters and vertices we can use the partition of the vertices to obtain a partition of the voters and vice versa. With arguments similar to the proof of Theorem 6.7 one can show that a set of vertices is a clique if and only if the corresponding profile is single-peaked consistent. □

Remark 6.9. The PARTITION INTO CLIQUES problem is NP-complete even when one is asked to partition the graph into three cliques. Consequently it follows from the proof of Theorem 6.8 that ADDITIONAL AXES SINGLE-PEAKED CONSISTENCY is NP-complete even for $k = 2$, i.e., for checking single-peaked consistency with two additional axes.

In the proofs of our next two results, we will provide reductions from the NP-complete MINIMUM RADIUS problem [82]. It is defined as follows:

MINIMUM RADIUS	
Instance:	A set of strings $S \subseteq \{0,1\}^\ell$ and a positive integer s.
Question:	Has S a radius of at most s, i.e., is there a string $\alpha \in \{0,1\}^\ell$ such that each string in S has a Hamming distance of at most s to α?

Theorem 6.10. *The* LOCAL CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY *problem is* NP*-complete.*

Proof. A MINIMUM RADIUS instance is given by $S \subseteq \{0,1\}^\ell$, the set of binary strings, and a positive integer s. Given a string β, let $\beta(i)$ denote the bit value at the i-th position in β. We are going to construct an *LCD* SINGLE-PEAKED CONSISTENCY instance. Each string in $S = \{\beta_1, \dots, \beta_n\}$ will correspond to a voter. Each bit of the strings corresponds to two candidates. In addition, we have $2\ell s + 2$ extra candidates. Consequently, we have $C = \{c_1^1, c_1^2, c_2^1, c_2^2, \dots, c_n^1, c_n^2, c'_1, \dots, c'_{\ell s+1}, c''_1, \dots, c''_{\ell s+1}\}$.

We define the preference profile with the help of two functions creating total orders.

$$f_0(a,b) = a \succ b \qquad\qquad f_1(a,b) = b \succ a$$

The vote V_k, for each $k \in \{1, \dots, \ell\}$, is of the form

$$c'_1 \ \dots \ c'_{\ell s+1} \ f_{\beta_k(1)}(c_1^1, c_1^2) \ f_{\beta_k(2)}(c_2^1, c_2^2) \ \dots \ f_{\beta_k(n)}(c_n^1, c_n^2) \ c''_1 \ \dots \ c''_{\ell s+1}.$$

The preference profile $\mathcal{P}$ is now defined as $(V_1, \dots, V_n, \overline{V_1}, \dots, \overline{V_n})$. We claim that $(C, \mathcal{P})$ is s-*LCD* single-peaked consistent if and only if S has a radius of at most s.

"$\Leftarrow$" Suppose that S has a radius of at most s, i.e., there is a string $\alpha \in \{0,1\}^\ell$ with Hamming distance at most s to each $\beta \in S$. We consider the following axis A:

$$c'_1 > \dots > c'_{\ell s+1} > f_{\alpha(1)}(c_1^1, c_1^2) > f_{\alpha(2)}(c_2^1, c_2^2) > \cdots f_{\alpha(n)}(c_n^1, c_n^2) > c''_1 > \dots > c''_{\ell s+1}.$$

We claim that $\mathcal{P}$ is single-peaked with respect to A after deleting at most s candidates in each vote. The deletions for vote V_k, $k \in \{1, \dots, \ell\}$, are the following: We delete candidate c_i^1 in V_k if and only if $\alpha(i) \neq \beta_k(i)$. The deletions in $\overline{V_k}$ are exactly the same as in V_k. These are at most s deletions since the Hamming distance between α and every $\beta \in S$ is at most s. After these deletions all votes are either subsequences of A or its reverse. Hence we obtain a single-peaked consistent profile.

"$\Rightarrow$" Let $\mathcal{P}'$ be the partial, single-peaked consistent profile that was obtained by deleting at most s candidates in each vote. First, note that some $c' \in \{c'_1, \dots, c'_{\ell s+1}\}$ has not been deleted in any vote since in total at most $\ell \cdot s$ many different candidates can be deleted. In the same way let $c'' \in \{c''_1, \dots, c''_{\ell s+1}\}$ be a candidate that has not been deleted in any vote. Now let us consider the profile $\mathcal{P}'[\{c', c'', c_i^1, c_i^2\}]$ for any $i \in \{1, \dots, n\}$. We claim that α, defined in the following way, has a Hamming distance of at most s to all bitstrings in S.

$$\alpha(k) = \begin{cases} 0 & \text{if } \mathcal{P}' \text{ contains the vote } c' \succ c_i^1 \succ c_i^2 \succ c'', \\ 1 & \text{if } \mathcal{P}' \text{ contains the vote } c' \succ c_i^2 \succ c_i^1 \succ c'', \\ 1 & \text{otherwise.} \end{cases}$$

First, observe that Case 1 and 2 cannot occur at the same time since then $\mathcal{P}'$ would not be single-peaked consistent. This is because $\mathcal{P}'[\{c', c'', c_i^1, c_i^2\}]$ also contains votes where c'' is ranked top and c' is ranked last and hence either $c' \succ c_i^1 \succ c_i^2 \succ c''$ or $c' \succ c_i^2 \succ c_i^1 \succ c''$ would not be single-peaked.

Let $\beta_j \in S$, $j \in \{1, \dots, n\}$. Note that if at any position i, $\beta_j(i) \neq \alpha(i)$ then either c_i^1 or c_i^2 had to be deleted in the vote V_j. Otherwise $\mathcal{P}'$ would not be single-peaked consistent. Hence the set $\{k \in \{1, \dots, \ell\} \mid \alpha(i) \neq \beta_j(i)\}$ cannot contain more than s elements because this would require more than s candidate deletions in the corresponding vote V_j. Hereby we have shown that the Hamming distance of α and β_j is at most s. $\square$

Theorem 6.11. LOCAL SWAPS SINGLE-PEAKED CONSISTENCY *is* NP*-complete.*

Proof. We use the same construction as in the proof of Theorem 6.10. It holds that $(C, \mathcal{P})$ is *s-LS* single-peaked consistent if and only if S has a radius of at most s. This can be shown similarly to the proof of Theorem 6.10 except that we swap elements instead of deleting them. $\square$

The following problem will be useful for showing NP-hardness of GLOBAL SWAPS SINGLE-PEAKED CONSISTENCY. Given two votes, V_x and V_y, let $\mathsf{swaps}(V_x, V_y)$ denote the minimum number of swaps of adjacent candidates needed to make V_x and V_y equal, i.e., $\mathsf{swaps}(V_x, V_y)$ is the Kendall-Tau distance of V_x and V_y.

KEMENY OPTIMAL AGGREGATION	
Instance:	An election $(C, \mathcal{P})$, with $\mathcal{P} = (V_1, \dots, V_n)$, and an integer s.
Question:	Is there a vote V^* over C such that $\sum_{1 \leq i \leq n} \mathsf{swaps}(V_i, V^*) \leq s$.

KEMENY OPTIMAL AGGREGATION was shown to be NP-hard in [22]. This result was strengthened in [65] to require only four voters.

Theorem 6.12. GLOBAL SWAPS SINGLE-PEAKED CONSISTENCY *is NP-complete.*

Proof. We show NP-hardness of this problem by reduction from KEMENY OPTIMAL AGGREGATION. Let a KEMENY OPTIMAL AGGREGATION instance be given by $(C, \mathcal{P})$ and an integer s. Furthermore, let $C = \{c_1, \ldots, c_m\}$, $\mathcal{P} = (V_1, \ldots, V_n)$ and let k be defined as $2s$.

We create a new election $(C', \mathcal{P}')$ with $C' = C \cup \{c_1^{\text{top}}, \ldots, c_{2k+1}^{\text{top}}, c_1^{\text{last}}, \ldots, c_{2k+1}^{\text{last}}\}$, i.e., $|C'| = m + 4k + 2$.

For each $i \in \{1, \ldots, m\}$ we create two votes V_i' and V_i'' as follows. The vote V_i' ranks c_1^{top} first, followed by c_2^{top}, c_3^{top}, …, c_{2k}^{top} and finally c_{2k+1}^{top}. Then it ranks the candidates in C in the same order as V_i does. Finally, it orders the candidates $c_1^{\text{last}} \cdots c_{2k+1}^{\text{last}}$ with descending preference, i.e., c_{2k+1}^{last} being the last ranked candidate. The preference profile $\mathcal{P}'$ is now defined as $(V_1', \overline{V_1'}, \ldots, V_n', \overline{V_n'})$.

We claim that $(C', \mathcal{P}')$ is k-*GS* single-peaked consistent if and only if $(C, \mathcal{P})$ and s are a yes-instance of the KEMENY OPTIMAL AGGREGATION problem.

"$\Rightarrow$" Suppose that $(C', \mathcal{P}')$ is k-*GS* single-peaked consistent. Therefore, one can obtain the profile $\mathcal{P}^S$ from $\mathcal{P}'$ by applying at most $k = 2s$ swaps such that $\mathcal{P}^S$ is single-peaked consistent with respect to an axis A. Since there are $2k + 1$ candidates in the set $\{c_1^{\text{top}}, \ldots, c_{2k+1}^{\text{top}}\}$ at least one of them must have remained in place in each vote. Analogously, the same holds for one of the candidates contained in the set $\{c_1^{\text{last}}, \ldots, c_{2k+1}^{\text{last}}\}$. Let c_{top} (resp. c_{last}) denote these two candidates. From Lemma 6.2 we know that $\mathcal{P}^S[\{c_{\text{top}}, c_1, \ldots, c_m, c_{\text{last}}\}]$ is single-peaked consistent as well. Observe that all primed votes in $\mathcal{P}^S[\{c_{\text{top}}, c_1, \ldots, c_m, c_{\text{last}}\}]$ have c_{top} as peak and c_{last} as last candidate, while in the double-primed votes c_{last} is top and c_{top} is the last ranked candidate. By Lemma 6.1 all primed votes in $\mathcal{P}^S[\{c_1, \ldots, c_m\}]$ must be ordered in the same way. We denote this ordering by V^*. The double-primed votes in $\mathcal{P}^S[\{c_1, \ldots, c_m\}]$, however, must be ordered according to the reverse of V^*. Notice that turning the primed votes into V^* requires the same number of swaps as turning the double-primed votes into the reverse of V^*. Therefore, $\frac{k}{2} = s$ swaps are sufficient to turn all primed votes into $\succ^*$. Taken together, V^* fulfills all properties to be a yes-instance of the KEMENY OPTIMAL AGGREGATION problem.

"$\Leftarrow$" Assume $(C, \mathcal{P})$ and s describe a yes-instance of the KEMENY OPTIMAL AGGREGATION problem. Then there is some common ordering V^*, which has in total a swap distance of $\leq s$ to all votes in $\mathcal{P}$. Let A^* be an axis ordering the candidates in C in the same way as V^* does. Then, $(C', \mathcal{P}')$ is k-*GS* single-peaked consistent with respect to the axis $c_1^{\text{top}} > c_2^{\text{top}} > \cdots > c_{2k+1}^{\text{top}} > [c_1, \ldots, c_n \text{ as ordered by } A^*] > c_1^{\text{last}} > c_2^{\text{last}} > \cdots > c_{2k+1}^{\text{last}}$. This is because all votes can be brought into the form $c_1^{\text{top}} \succ c_2^{\text{top}} \succ \cdots \succ c_{2k+1}^{\text{top}} \succ [c_1, \ldots, c_n \text{ as ordered by } V^*] \succ c_1^{\text{last}} \succ c_2^{\text{last}} \succ \cdots \succ c_{2k+1}^{\text{last}}$ or its reverse by using at most $k = 2s$ swaps – s swaps for the primed votes and s swaps for the double-primed votes. □

Remark 6.13. Since KEMENY OPTIMAL AGGREGATION is NP-complete [65] even with only four voters, it follows from the proof of Theorem 6.12 that GLOBAL SWAPS SINGLE-PEAKED CONSISTENCY is NP-complete even for eight voters.

6.4.2 A polynomial time algorithm for CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY

In contrast to the previous hardness results, we are able to show that CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY can be decided in polynomial time. The algorithm builds upon the $\mathcal{O}(n \cdot m)$ time algorithm for testing single-peaked consistency by Escoffier, Lang, and Ötztürk [71]. Since we make some modifications to the algorithm and also for the sake of completeness we present it here as well. For the remainder of this section let $(C, \mathcal{P})$ be an election with n voters and $C = \{c_1, \ldots, c_m\}$.

The single-peaked consistency algorithm. This algorithm is a modified version of the algorithm by Escoffier, Lang, and Ötztürk [71]. We start by giving three fundamental definitions that we use to state the algorithm.

Definition 6.9. *$L(\mathcal{P}, C')$ is the set of last ranked candidates in $\mathcal{P}[C']$.*

Definition 6.10. *A* partial axis *A is a total order of a subset of the candidates in C. Let $cand(A)$ denote the candidates that are ordered by A. Consequently, any partial axis A is an axis over $cand(A)$. By the* cardinality of a partial axis A *we mean* $|cand(A)|$.

Definition 6.11. *An* incomplete axis *is a partial axis with a marked position that indicates where further elements may be added. We denote this position by a star symbol, e.g., the incomplete axis $c_1 > c_2 > \star > c_3$ allows additional candidates to be added right of c_2 and left of c_3. The* boundary *of an incomplete axis A,* $\mathsf{boundary}(A)$*, is a quadruple consisting of the two candidates left of the star and the two candidates right of the star, e.g.,* $\mathsf{boundary}(c_1 > c_2 > \star > c_3 > c_4 > c_5) = (c_1, c_2, c_3, c_4)$. *If only one or no candidates exist left/right of the star, the corresponding entry in the quadruple is ϵ, e.g.,* $\mathsf{boundary}(c_1 > \star) = (\epsilon, c_1, \epsilon, \epsilon)$.

Given an incomplete axis A and a candidate set C, an axis A' extends A if A' can be constructed from A' by adding elements left or right of the $\star$ symbol.

The algorithm by Escoffier, Lang and Ötztürk proceeds iteratively by placing the last ranked candidates that have not yet been placed. Let C' be the set of candidates that have not yet been positioned on the (incomplete) axis A. The algorithm checks what kinds of constraints follow from each vote. If these constraints do not contradict each other, the set of last ranked candidates $L(\mathcal{P}, C')$ is placed. We denote this procedure with $\mathsf{place}(A, X)$ where $X = L(\mathcal{P}, C')$. The procedure $\mathsf{place}(A, X)$ returns either a new incomplete axis (extending A by the candidates in X) or the value `INCONSISTENT`. The algorithm repeatedly invokes place until all elements have been placed or a contradiction has been found.

Now we would like to describe $\mathsf{place}(A, X)$ in detail since it is used also by our candidate deletion algorithm. Let $\mathsf{boundary}(A) = (b_1', b_1, b_2, b_2')$, i.e., $A = \cdots < b_1' < b_1 < \star < b_2 < b_2' < \cdots$. If a condition contains a boundary element and this element is ϵ (i.e., it does not exist), the corresponding constraint is not valid. The following cases are considered for each vote $\prec_k$, $k \in \{1, \ldots, n\}$ and thus we obtain constraints on all possible placements of X.

Case 1. $|L(\mathcal{P}, C')| \geq 3$. There are three or more candidates that would have to be placed at the positions next to b_1 and b_2. Since this is not possible, $\mathcal{P}$ is not single-peaked consistent.

Case 2. $L(\mathcal{P}, C') = \{x_1, x_2\}$. The candidates x_1 and x_2 have to be placed at the positions next to b_1 and next to b_2.

a) $x_1 \prec_k b_1$ and $x_1 \prec_k b_2$: In this case x_1 can be placed neither left nor right and thus place returns `INCONSISTENT`.

b) $b_1 \prec_k x_1$ and $b_2 \prec_k x_1$: There are no constraints for x_1 that follow from vote V_k.

c) $b_1 \prec_k x_1 \prec_k b_2 \prec_k x_2$: x_1 has to be placed next to b_1 and therefore x_2 is placed next to b_2.

d) $b_1 \prec_k x_1 \prec_k x_2 \prec_k b_2$: x_1 has to be placed next to b_1 and therefore x_2 has to be placed next to b_2.

All these rules are also applicable if b_1 and b_2 are interchanged and also if x_1 and x_2 are interchanged.

Case 3. $L(\mathcal{P}, C') = \{x\}$. The candidate x has to be placed either at the position next b_1 or b_2.

a) $x \prec_k b_1$ and $x \prec_k b_2$: In this case x can be placed neither left nor right and thus place returns `INCONSISTENT`.

b) $b_1 \prec_k x$ and $b_2 \prec_k x$: There are no constraints for x.

c) $b_1 \prec_k x \prec_k b_2$: x has to be placed next to b_1.

d) $b_2 \prec_k x \prec_k b_1$: x has to be placed next to b_2.

In addition to these three cases, the following constraints are applicable independent of the cardinality of $L(\mathcal{P}, C')$. Let $x \in L(\mathcal{P}, C')$.

- If $b_1' \succ_k b_1$ and $x \succ_k b_1$, then x can be placed neither left nor right.
- If $b_2' \succ_k b_2$ and $x \succ_k b_2$, then x can be placed neither left nor right.

For each vote $\prec_k$, these case distinctions yield constraints on placing the candidates in X. If there is a way to place the candidates in X that is compatible with every vote, place(A, X) is successful and returns the new incomplete axis. (If there is more than one possibility to place X, we choose arbitrarily.) Otherwise the value `INCONSISTENT` is returned. To simplify the notation, we define place$(A, \emptyset)$ to return A.

The following lemma is the main reason why we can employ dynamic programming in our algorithm for deciding the CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY problem.

Lemma 6.14. *Let A be an incomplete axis and let $X \subseteq C$ contain one or two candidates not yet placed on A. If $\mathcal{P}$ is single-peaked with respect to an axis A' that extends A, then it is single-peaked with respect to an extension of the axis returned by place(A, X). If place(A, X) returns `INCONSISTENT`, then there is no axis A' that extends A such that $\mathcal{P}$ is single-peaked with respect to A'.*

Proof. This lemma follows from the correctness proof of the single-peaked consistency algorithm [71] since the place procedure performs the same steps as this algorithm does. The main difference is that the single-peaked consistency algorithm places all remaining candidates at once as soon there is at most one possibility left. The place procedure continues to place one or two candidates at a time even in that case. □

Observation 6.15. The $\mathsf{place}(A, X)$ procedure places the candidates in X only considering $\mathsf{boundary}(A)$ and does not depend on the full incomplete axis A.

The candidate deletion algorithm. Observation 6.15 states that the (at most) four boundary candidates of an incomplete axis fully determine whether and which further candidates can be placed on the axis. The main idea of our algorithm is to store only incomplete axes that differ in these four candidates, i.e., only incomplete axes with differing boundaries. If two axes with the same boundary are considered, we take the incomplete axis with the larger cardinality. This strategy allows for a dynamic programming approach.

Our algorithm resembles the previously described single-peaked consistency algorithm in that it places last ranked candidates first. However, since we are allowed to delete candidates, our algorithm does not terminate if at some point three or more last ranked candidates are encountered (cf. Case 1 in the single-peaked consistency algorithm). Nevertheless, our algorithm utilizes the place procedure and thus can place at most two candidates in each step. To this end, we define a sequence $L_1, \ldots, L_m$ of sets, each of which contains a disjoint subset of candidates. Our algorithm places the candidates in L_1 (or a subset of L_1) first, then (a subset of) those in L_2 and so on.

The sequence $L_1, \ldots, L_m$ is defined as follows:

$$L_i = L\left(\mathcal{P}, C \setminus (L_1 \cup \cdots \cup L_{i-1})\right). \tag{6.4.1}$$

Note that some L_i's might be empty and that $\bigcup_{i\in[m]} L_i = C$.

Now, we describe the algorithm. Refer to Algorithm 3 for an overview. The main data structure is an array $\mathcal{S}$ containing incomplete axes. Each position in this array is uniquely described by a quadruple of candidates. Consequently, the array has size m^4. Each incomplete axis is stored at the position that corresponds to the boundary of this axis.

We start with the array $\mathcal{S}$ containing only the empty incomplete axis $\star$. (Recall that $\star$ marks the position where new candidates can be added to the axis.) Now, the candidates in L_1 are placed. We make a copy of $\mathcal{S}$ called $\mathcal{S}_{\text{new}}$. Then, we use the place procedure to place the candidates in L_1 on the empty axis. Note that at most two of the candidates in L_1 can be placed, since otherwise we would create two peaks. Thus, we consider every subset of L_1 of size 0, 1 or 2. This gives rise to new incomplete axes. These axes are stored in $\mathcal{S}_{\text{new}}$, a copy of S. After all possible axes are created, we replace $\mathcal{S}$ with $\mathcal{S}_{\text{new}}$.

We continue by placing the candidates in L_2. Again, we copy $\mathcal{S}$ to $\mathcal{S}_{\text{new}}$. For every incomplete axis A in $\mathcal{S}$, we place any subset of L_2 with size 0, 1 or 2 on A – and again create new incomplete axes. These axes are stored in $\mathcal{S}_{\text{new}}$. At this point, it might be that $\mathcal{S}_{\text{new}}$ already contains an axis with the same boundary. In this case, we keep the axis with a larger cardinality.

We repeat this procedure until the candidates in L_m are placed as well. We say that one axis A_1 is a representative of axis A_2 if $\mathsf{boundary}(A_1) = \mathsf{boundary}(A_2)$ and $|A_1| \geq |A_2|$. The set $\mathcal{S}$ now contains one representative axis for every possible incomplete axis. Consequently, $\mathcal{S}$ contains a cardinality maximal axis and thus yields the minimum number of candidates that have to be deleted to make $\mathcal{P}$ single-peaked.

Algorithm 3: Polynomial time algorithm for k-*CD* single-peaked consistency – Theorem 6.16

```
S ← { ⋆ }                      // S contains the empty incomplete axis
Choose L_1, ..., L_m according to Equation 6.4.1
for i = 1 ... m do
    S_new ← S
    foreach X ⊆ L_i with 0 ≤ |X| ≤ 2 do
        foreach incomplete axis A ∈ S do
            A_new ← place(A, X)
            if A_new ≠ INCONSISTENT then
                (c_1, c_2, c_3, c_4) ← boundary(A_new)
                if S[c_1, c_2, c_3, c_4] is empty then
                    S[c_1, c_2, c_3, c_4] ← A_new
                else
                    if |cand(A_new)| > |cand(S[c_1, c_2, c_3, c_4])| then
                        S[c_1, c_2, c_3, c_4] ← A_new
    S ← S_new
return an axis A ∈ S with maximum |cand(A)|
```

Theorem 6.16. CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY *can be solved in time* $\mathcal{O}(n \cdot m^6)$.

Proof. The runtime bound can be seen as follows. Clearly, $L_1, \dots, L_m$ can be computed in $\mathcal{O}(m \cdot n)$ time. The algorithm places the candidates L_1 first, then L_2, and so on. We consider $\mathcal{O}(|L_i|^2)$ many subsets of L_i (those of cardinality at most 2). Thus, the place procedure is executed at most $|L_1|^2 \cdot |L_2|^2 \cdot \dots \cdot |L_m|^2$ times. This number can be bounded by $\mathcal{O}(m^2)$. Since the array $\mathcal{S}$ has size at most m^4 and since place has a runtime of $\mathcal{O}(n)$, we require in total $\mathcal{O}(n \cdot m^6)$ time. $\square$

6.5 Complexity of Nearly Single-Peaked Evaluation

In the previous section we have analyzed the computational complexity of the X SINGLE-PEAKED CONSISTENCY problem. We now turn to the computational complexity of the related X SINGLE-PEAKED EVALUATION problem, where the axis is also given in the input. Because of this additional information, the X SINGLE-PEAKED EVALUATION problem becomes tractable where the X SINGLE-PEAKED CONSISTENCY problem was NP-complete.

Proposition 6.17. VOTER DELETION SINGLE-PEAKED EVALUATION *can be solved in time* $\mathcal{O}(n \cdot m)$.

Proof. This result is trivial due to the fact that whenever a vote is not single-peaked consistent with respect to axis A we have to delete it. If at most k votes have to be deleted, we know that the profile is k-voter deletion single-peaked consistent with respect to A. □

Proposition 6.18. CANDIDATE DELETION SINGLE-PEAKED EVALUATION *can be solved in time* $\mathcal{O}(n \cdot m^6)$.

Proof. We employ the algorithm for solving CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY, Algorithm 3. The only necessary modification is to change the place procedure in such a way that only placements compatible with the given axis are allowed. □

Proposition 6.19. ADDITIONAL AXES SINGLE-PEAKED EVALUATION *can be solved in time* $\mathcal{O}(k \cdot n \cdot m)$.

Proof. Evaluation is also trivial for the additional axes distance measure. It suffices to check each of the $k + 1$ axes for single peaked-consistency, which can be done in $\mathcal{O}(n \cdot m)$. □

Theorem 6.20. LOCAL CANDIDATE DELETION SINGLE-PEAKED EVALUATION *can be solved in time* $\mathcal{O}(n \cdot m^2 \cdot \log m)$.

Proof. For every vote $V \in \mathcal{P}$ we have to find the minimum number of candidates that have to be deleted. We iterate over all candidates; let p be this candidate and V the vote under consideration. We first consider V restricted to p and the candidates left of it. We have to a find subsequence of maximum length that is increasing with respect to the axis A and increasing with respect to this restricted vote. We delete the candidates not contained in this subsequence. Then, we consider the vote V restricted to the candidates right of p. This time, we search for a subsequence of maximum length that is increasing with respect to the axis A and decreasing with respect to this restricted vote. Again, we delete the candidates not contained in this subsequence. In this way, we obtain a selection of candidates C' such that $V[C']$ is single-peaked with respect to A.

We repeat this procedure for every vote. This yields a local deletion distance for every choice of p. The smallest of these distances is optimal.

Since computing a longest increasing subsequence can be done for sequences of length m in time $\mathcal{O}(m \cdot \log m)$ [129], we obtain a total runtime of $\mathcal{O}(n \cdot m^2 \cdot \log m)$. □

Theorem 6.21. *The* GLOBAL SWAPS SINGLE-PEAKED EVALUATION *and* LOCAL SWAPS SINGLE-PEAKED EVALUATION *problem can be solved in time* $\mathcal{O}(n \cdot m^2)$.

Proof. Both algorithms rely on the minswaps(V, A) procedure, which computes the minimal number of swaps required to make vote V single-peaked with respect to A. Let us first describe how this procedure is used and later on give a precise description of minswaps. To solve GLOBAL SWAPS SINGLE-PEAKED EVALUATION it suffices to execute for each V in $\mathcal{P}$ the procedure minswaps(V, A) and sum over all returned values. If the sum does not exceed the limit k we know that the profile is k-global swaps single-peaked consistent with respect to A. For the LOCAL SWAPS SINGLE-PEAKED EVALUATION the procedure is similar. Here, we check whether for every V in $\mathcal{P}$, minswaps$(V, A) \leq k$.

Let us now describe how $\mathsf{minswaps}(V, A)$ works. The procedure is depicted in Algorithm 4. It is based on the observation that one of the two outermost candidates on A have to be ranked last in V. These candidates are a and b in the algorithm. It is optimal to swap the lower one of these two candidates to the last position. The function $\mathsf{bottomdist}(c, V)$, which is used in the algorithm, computes the number of swaps required to swap candidate c in vote V to the last position.

After either a or b have been swapped to the last position, we repeat these steps with both the vote and the axis restricted to those candidates that have not been swapped to the last position so far. In this way we obtain a vote with a minimal number of swaps that is single-peaked. Since the runtime of the procedure $\mathsf{minswaps}$ can be bounded by $\mathcal{O}(m)$, GLOBAL SWAPS SINGLE-PEAKED EVALUATION as well as LOCAL SWAPS SINGLE-PEAKED EVALUATION can be solved in time $\mathcal{O}(n \cdot m)$. □

Algorithm 4: Procedure $\mathsf{minswaps}(V, A)$ used in Theorem 6.21

```
s ← 0
C′ ← C
while C′ ≠ ∅ do
    a ← rightmost candidate on A[C].
    b ← leftmost candidate on A[C].
    s_a ← bottomdist(a, V[C′])
    s_b ← bottomdist(b, V[C′])
    if s_a < s_b then
        s ← s + s_a
        C ← C \ {a}
    else
        s ← s + s_b
        C ← C \ {b}
return s
```

	SP-Consistency	SP-Evaluation
k-Voter Deletion	NP-c (Thm. 6.7)	in P (Prop. 6.17)
k-Candidate Deletion	in P (Thm. 6.16)	in P (Prop. 6.18)
k-Local Candidate Deletion	NP-c (Thm. 6.10)	in P (Thm. 6.20)
k-Additional Axes	NP-c (Thm. 6.8)	in P (Prop. 6.19)
k-Global Swaps	NP-c (Thm. 6.12)	in P (Thm. 6.21)
k-Local Swaps	NP-c (Thm. 6.11)	in P (Thm. 6.21)
k-Candidate Partition	open	open

Table 6.2: Complexity results for different notions of nearly single-peakedness

6.6 Summary

In this chapter we have studied seven notions measuring nearly single-peakedness. Three of them are novel: the local candidate deletion distance, the global swaps distance and the candidate partition distance; the other four have already been defined or suggested in the literature. We have drawn a complete picture of the relations between all the notions of nearly single-peakedness discussed in this chapter (cf. Figure 6.1 and Table 6.1). For five notions we have shown that deciding single-peaked consistency is NP-complete and for k-candidate deletion we have presented a polynomial time algorithm. For the simpler single-peaked evaluation problem, where an axis is given as part of the input, we found polynomial-time algorithms for all these six cases. We refer the reader to Table 6.2 for an overview.

Finally, we would like to remark that these notions of distance are not only applicable to single-peakedness. Most of the notions are immediately applicable to other domain restrictions such as the single-crossing restriction. Only k-additional axes explicitly concerns axes and is thus not trivially applicable to arbitrary domain restrictions. However, if we view k-additional axes as the number of partitions of voters such that each corresponding set of votes satisfies the domain restriction, this definition is equivalent and applicable to arbitrary domain restrictions.

CHAPTER 7

Nearly Structured Preferences: Efficient Detection

This chapter is based on the publication *On detecting nearly structured preference profiles* [68], a joint work with Edith Elkind.

In the previous chapter we have seen that it is often NP-hard to determine whether a preference profile is close to single-peakedness. It is then natural to ask whether these hardness results can be circumvented using approximation algorithms and/or parameterized algorithms. The main contribution in this chapter is answering this question in the affirmative for the voter deletion distance and the candidate deletion distance. Our results do not only apply to the single-peaked domain but to a large family of restricted domains. Specifically, our results apply to any restricted domain that can be characterized in terms of *forbidden configurations* (see Section 7.1); this includes all domains discussed by Bredereck et al. [44]. For any such domain $\mathcal{D}$, we present approximation algorithms for the problem of finding the smallest number of voters/candidates to delete in order to obtain an election in $\mathcal{D}$. The approximation ratio on our algorithm is determined by the size of the largest forbidden configuration used to characterize $\mathcal{D}$, which is typically a small integer. Our algorithm proceeds by reducing our problem to the classic HITTING SET problem. For the voter deletion distance and several restricted domains (including, notably, the single-peaked domain), we can improve the approximation ratio of our algorithm by using a more elaborate reduction to HITTING SET; this approach results in a 2-approximation algorithm. We then show that this result is optimal subject to a plausible complexity-theoretic assumption.

Our reduction to HITTING SET also allows us to use parameterized algorithms for this problem, resulting in fpt algorithms for our problem. For a summary of approximation and fpt results, we refer the reader to the summary, Section 7.7.

For voter deletion, we also consider the setting where we need to delete more than half of the voters. In this case, from the approximation algorithms perspective, it is more natural to focus on approximating the number of surviving voters. We show that this problem is W[1]-complete, and cannot be approximated within $n^{1-\epsilon}$ unless $\mathsf{P} \neq \mathsf{NP}$.

7.1 Configurations

A *condition* on a set of variables $X = \{x_1, \dots, x_t\}$ is a Boolean formula with pairwise comparisons of $x_1, \dots, x_t$ as atoms. For instance, $\phi : x_1 > x_2 \wedge x_3 > x_4$ (or short: $\phi : x_1x_2 \wedge x_3x_4$) is a condition on $\{x_1, x_2, x_3, x_4\}$. A *configuration* is a set of conditions $\Phi = \{\phi_1, \dots, \phi_s\}$, where all ϕ_i, $i \in [s]$, are conditions over a common set of variables. We denote by $s(\Phi)$ the number of conditions in Φ and by $X(\Phi)$ the set of variables that occur in Φ; also, we write $t(\Phi) = |X(\Phi)|$. We refer to a configuration Φ with $s(\Phi) = s$, $t(\Phi) = t$ as an *(s,t)-configuration*. Let $\|\Phi\|$ denote the input size (required space) of a configuration. Since we only consider configurations on a variable set of small constant size, the representation details do not affect the complexity of our algorithms.

The following definition plays a central role in this chapter as well as in Chapter 9.

Definition 7.1. *Given an injective function $\xi : X \to C$ and a condition ϕ over X, let $\xi(\phi)$ denote the Boolean formula obtained by replacing all variables in ϕ according to ξ. A vote V over C* fulfills ϕ with respect to ξ *(and write $V \models_\xi \phi$) if V is a model for $\xi(\phi)$. An election $E = (C, \mathcal{P})$ is said to* contain a configuration $\Phi = \{\phi_1, \dots, \phi_s\}$ *with $X(\Phi) = X$ if there exists an injective function $\xi : X \to C$ and s distinct votes $V_{i_1}, \dots, V_{i_s} \in \mathcal{P}$ such that $V_{i_j} \models_\xi \phi_j$ for all $j \in [s]$.*

Example 7.1. Consider an election $E = (C, \mathcal{P})$, where $C = \{c_1, c_2, c_3, c_4\}$, $\mathcal{P} = (V_1, V_2)$, $V_1 : c_1c_2c_3c_4$, $V_2 : c_4c_1c_2c_3$, and a configuration $\Phi = \{\phi_1, \phi_2\}$, where $\phi_1 : abc$, $\phi_2 : bca$. Then E contains Φ. Indeed, if we set $\xi(a) = c_4$, $\xi(b) = c_1$, $\xi(c) = c_2$, $V_{i_1} = V_2$, $V_{i_2} = V_1$, we get $V_{i_1} \models_\xi \phi_1$, $V_{i_2} \models_\xi \phi_2$. $\dashv$

We will now introduce five configurations that will play an important role in this chapter.

Definition 7.2. *The* α-configuration *is a $(2,4)$-configuration Φ_α with conditions*

$$\phi_1 : abc \wedge db, \qquad \phi_2 : cba \wedge db.$$

The $\bar{\alpha}$-configuration *is a $(2,4)$-configuration $\Phi_{\bar{\alpha}}$ with conditions*

$$\phi_1 : abc \wedge bd, \qquad \phi_2 : cba \wedge bd.$$

The β-configuration *is a $(2,4)$-configuration Φ_β with conditions*

$$\phi_1 : abcd, \qquad \phi_2 : bdac.$$

The γ-configuration *is a $(3,6)$-configuration Φ_γ with conditions*

$$\phi_1 : ba \wedge cd \wedge ef, \qquad \phi_2 : ab \wedge dc \wedge ef, \qquad \phi_3 : ab \wedge cd \wedge fe$$

The δ-configuration *is a $(4,4)$-configuration Φ_δ with conditions*

$$\phi_1 : ab \wedge cd, \quad \phi_2 : ab \wedge dc,$$
$$\phi_3 : ba \wedge cd, \quad \phi_4 : ba \wedge dc$$

The worst-diverse configuration *is a* $(3,3)$*-configuration* Φ_W *with conditions*

$$\phi_1 : ac \wedge bc, \quad \phi_2 : ab \wedge cb, \quad \phi_3 : ba \wedge ca.$$

The best-diverse configuration *is a* $(3,3)$*-configuration* Φ_B *with conditions*

$$\phi_1 : ab \wedge ac, \quad \phi_2 : ba \wedge bc, \quad \phi_3 : ca \wedge cb.$$

The medium-diverse configuration *is a* $(3,3)$*-configuration* Φ_M *with conditions*

$$\phi_1 : bac \vee cab, \quad \phi_2 : abc \vee cba, \quad \phi_3 : acb \vee bca.$$

The value-diverse configuration *is a* $(3,3)$*-configuration* Φ_C *with conditions*

$$\phi_1 : abc, \qquad \phi_2 : bca, \qquad \phi_3 : cab.$$

An election is said to be *worst-restricted* if it does not contain Φ_W; *best-restricted*, *medium-restricted*, and *value-restricted* elections are defined similarly.

We will now formulate two simple conditions on configurations.

Definition 7.3. *A configuration* Φ *is* exact *if every preference order over* $X(\Phi)$ *fulfills at most one condition in* Φ*. Further,* Φ *is* partitioning *if every preference order over* $X(\Phi)$ *fulfills exactly one condition in* Φ*.*

Observe that Φ_α, Φ_W, Φ_B, Φ_M, and Φ_C are exact configurations; further, Φ_W, Φ_B, and Φ_M are partitioning, but Φ_α and Φ_C are not.

The notion of partitioning configurations will play an important role in Section 7.3. We will now describe an efficient algorithm for checking whether an election E contains an exact configuration Φ.

Proposition 7.1. *Given an exact configuration* Φ *with* $s(\Phi) = s$*,* $t(\Phi) = t$ *and an election* $E = (C, \mathcal{P})$ *with* $|C| = m$*,* $|\mathcal{P}| = n$*, we can detect whether* E *contains* Φ *in time* $\mathcal{O}(\|\Phi\| n m^t)$*.*

Proof. We can go over all ordered t-tuples of elements of C. Each such tuple can be interpreted as a mapping ξ from $X = X(\Phi)$ to C. For each such mapping, we set $\Phi' \leftarrow \Phi$ and go over the votes in $\mathcal{P}$ one by one. For each vote $V \in \mathcal{P}$, we check whether $V \models_\xi \phi_i$ for some $\phi_i \in \Phi'$; this can be done in time $\mathcal{O}(\|\Phi\|)$. Note that, since Φ is exact, there can be at most one such condition. If $V \models_\xi \phi_i$, we remove ϕ_i from Φ', and repeat this process with the next vote in $\mathcal{P}$. If Φ' becomes empty, we return "yes" and stop. If all votes in $\mathcal{P}$ have been processed, but Φ' remains non-empty, we move on to the next mapping $\xi : X \to C$ (and reset $\Phi' \leftarrow \Phi$). If we have enumerated all mappings $\xi : X \to C$, we stop and output "no". The correctness of this algorithm and the bound on its running time are immediate. □

If Φ is not exact, the algorithm described in the proof of Proposition 7.1 may fail to work correctly. However, by considering all mappings $\xi : X \to C$ and all ordered s-tuples of voters in $\mathcal{P}$, we can check whether E contains Φ in time $\mathcal{O}(\|\Phi\| n^s m^t)$.

We say that a preference domain $\mathcal{D}$ is *characterized by a set of forbidden configurations* $\Gamma = \{\Phi_1, \dots, \Phi_\gamma\}$ if for every election E we have $E \in \mathcal{D}$ if and only if E does not contain any of the configurations in Γ.

By definition, the domains of worst-restricted, best-restricted, medium-restricted, and value-restricted elections can be characterized by sets of forbidden configurations that consist of a single $(3, 3)$-configuration each. Moreover, the following results are known.

- The domain of single-peaked preferences is characterized by the set of forbidden configurations $\{\Phi_\alpha, \Phi_W\}$ [15].
- The domain of single-crossing preferences is characterized by the set of forbidden configurations $\{\Phi_\gamma, \Phi_\delta\}$ [45].
- The domain of single-caved preferences is characterized by the set of forbidden configurations $\{\Phi_{\bar{\alpha}}, \Phi_B\}$ [15].
- The domain of group-separable preferences is characterized by the set of forbidden configurations $\{\Phi_\beta, \Phi_M\}$. [15].

Observe that each of the configurations $\Phi_{\bar{\alpha}}$, Φ_β, Φ_γ, Φ_δ is exact. We set $\Gamma_W = \{\Phi_W\}$, $\Gamma_B = \{\Phi_B\}$, $\Gamma_M = \{\Phi_M\}$, $\Gamma_C = \{\Phi_C\}$, $\Gamma_{sp} = \{\Phi_\alpha, \Phi_W\}$, $\Gamma_{scv} = \{\Phi_{\bar{\alpha}}, \Phi_B\}$, $\Gamma_{sc} = \{\Phi_\gamma, \Phi_\delta\}$, $\Gamma_{gs} = \{\Phi_\beta, \Phi_M\}$.

We will now define the two families of computational problems that will be the focus of this chapter. Both families take a set of configurations Γ as an argument, determining which domain restriction is considered.

Γ-VDEL	
Instance:	An election $E = (C, \mathcal{P})$.
Question:	Find the smallest k such that for some $\mathcal{P}' \subseteq \mathcal{P}$ with $\lvert\mathcal{P}'\rvert = k$ the election $(C, \mathcal{P} \setminus \mathcal{P}')$ contains no configurations from Γ.

Γ-CDEL	
Instance:	An election $E = (C, \mathcal{P})$.
Question:	Find the smallest k such that for some $C' \subseteq C$ with $\lvert C'\rvert = k$ the restriction of E to $C \setminus C'$ contains no configurations from Γ.

Note that Γ_{sp}-VDEL is the corresponding function problem of VOTER DELETION SINGLE-PEAKED CONSISTENCY, which we have studied in the previous chapter; that is, Γ_{sp}-VDEL asks for a set of voters whereas VOTER DELETION SINGLE-PEAKED CONSISTENCY asks whether such a set exists. For Γ_{sp}-CDEL and CANDIDATE DELETION SINGLE-PEAKED CONSISTENCY the same relation holds.

7.2 A Simple Conversion to Hitting Set

In this section, we describe a straightforward transformation from Γ-VDEL and Γ-CDEL to the classic HITTING SET problem, which is defined as follows.

d-HITTING SET	
Instance:	A finite set A and a collection $\mathcal{T}$ of subsets of A, where $|S| \leq d$ for all $S \in \mathcal{T}$.
Question:	Find the smallest k such that there is a set $A' \subseteq A$ with $|A'| = k$ satisfying $A' \cap S \neq \emptyset$ for each $S \in \mathcal{T}$.

Theorem 7.2. *Let Γ be a set of exact configurations, let $\|\Gamma\| = \sum_{\Phi\in\Gamma} \|\Phi\|$, and let $s = \max_{\Phi\in\Gamma} s(\Phi)$, $t = \max_{\Phi\in\Gamma} t(\Phi)$. Then an instance $E = (C, \mathcal{P})$ of Γ-VDEL (respectively, Γ-CDEL) with $|C| = m$, $|\mathcal{P}| = n$ can be reduced to an instance $(A, \mathcal{T})$ of d-HITTING SET with $d = s$ (respectively, $d = t$) in time $\mathcal{O}(\|\Gamma\|nm^t)$ so that the optimal number of voters (respectively, candidates) to delete in E equals the optimal size of the hitting set for $(A, \mathcal{T})$.*

Proof. We first consider Γ-VDEL. Given an election $E = (C, \mathcal{P})$, we set $A = \mathcal{P}$. Further, for each occurrence of a forbidden configuration from Γ in E we add the corresponding set of voters to $\mathcal{T}$. We obtain an instance of d-HITTING SET with $d = \max_{\Phi\in\Gamma} s(\Phi)$. For Γ-CDEL, the reduction is similar: we set $A = C$, and the sets in $\mathcal{T}$ correspond to sets of candidates in occurrences of configurations from Γ in E.

Let $(A, \mathcal{T})$ be the instance of d-HITTING SET produced by our reduction. Suppose that we can eliminate all occurrences of the configurations in Γ from E by deleting a set of voters $\mathcal{P}' \subseteq \mathcal{P}$. Then $\mathcal{P}'$ intersects every set in $\mathcal{T}$, so $(A, \mathcal{T})$ admits a hitting set of size $|\mathcal{P}'|$. Conversely, if A' is a hitting set for $(A, \mathcal{T})$, then by deleting the corresponding voters from $\mathcal{P}$ we ensure that our election contains no configurations in Γ. A similar argument works for Γ-CDEL.

To implement this reduction, we go over all configurations in Γ, and, for each configuration Φ, detect all occurrences of Φ in E using a modification of the algorithm described in Proposition 7.1. This establishes the bound on the running time of our reduction. □

This simple conversion enables us to use the techniques developed for d-HITTING SET in order to solve Γ-VDEL and Γ-CDEL whenever all configurations in Γ are exact and $t = \max_{\Phi\in\Gamma} t(\Phi)$ is bounded by a small constant; this is the case for all sets of forbidden configurations considered in this thesis. These techniques include, in particular, approximation algorithms and fpt algorithms for d-HITTING SET. However, the running time and/or solution quality of these algorithms often depends on the value of d. Thus, it would be desirable to have a reduction that produces an instance of d-HITTING SET with a smaller value of d. We will now see that this is indeed possible for Γ-VDEL, for several important sets of forbidden configurations Γ, including the one that characterizes single-peaked preferences.

7.3 An Improved Conversion to Hitting Set

Our improved conversion from Γ-VDEL to d-HITTING SET relies on the notion of a partitioning configuration (Definition 7.3). Given an election $E = (C, \mathcal{P})$ and a mapping $\xi : X \to C$, a partitioning (s,t)-configuration $\Phi = \{\phi_1, \dots, \phi_s\}$ with $X(\Phi) = X$ induces a partition of $\mathcal{P}$ into s sets $\mathcal{P}_1^\xi, \dots, \mathcal{P}_s^\xi$, where $\mathcal{P}_i^\xi = \{V \in \mathcal{P} \mid V \models_\xi \phi_i\}$ for each $i \in [s]$. Using this observation, for Γ-VDEL we can strengthen Theorem 7.2 as follows.

Theorem 7.3. *Let Γ be a set of exact configurations, let $\|\Gamma\| = \sum_{\Phi\in\Gamma} \|\Phi\|$, and let $s = \max_{\Phi\in\Gamma} s(\Phi)$, $t = \max_{\Phi\in\Gamma} t(\Phi)$, where $s \geq 3$. Suppose also that Γ contains exactly one configuration Φ^+ with $s(\Phi^+) = s$, and this configuration is partitioning. Then, given an instance $E = (C, \mathcal{P})$ of Γ-VDEL with $|C| = m$, $|\mathcal{P}| = n$, where the optimal solution size is less than $\frac{n}{s-1}$, we can construct s instances of $(s-1)$-HITTING SET in time $\mathcal{O}(\|\Gamma\| nm^t)$ so that the optimal number of voters to delete in E equals $\min_{i\in[s]} |A_i|$, where A_i is an optimal hitting set for the i-th instance.*

Proof. We construct s instances of $(s-1)$-HITTING SET, denoted by $(A, \mathcal{T}_1), (A, \mathcal{T}_2), \dots, (A, \mathcal{T}_s)$. We set $A = \mathcal{P}$. The sets $\mathcal{T}_1, \dots, \mathcal{T}_s$ are constructed in three steps.

Step 1. Let $\Phi^+ = \{\phi_1, \dots, \phi_s\}$ be the unique configuration in Γ with $s(\Phi^+) = s$; let $X = X(\Phi^+)$. As explained above, for every mapping $\xi : X \to C$, Φ^+ defines a partition of $\mathcal{P}$ into sets of votes $\mathcal{P}_1^\xi, \dots, \mathcal{P}_s^\xi$. Pick a mapping ξ that maximizes the size of the smallest set in $\{\mathcal{P}_1^\xi, \dots, \mathcal{P}_s^\xi\}$. For each $i \in [s]$, initialize $\mathcal{T}_i$ by setting $\mathcal{T}_i = \Big\{\{V\} \mid V \in \mathcal{P}_i^\xi\Big\}$.

Step 2. Let $\mathcal{P}_{-i} = \mathcal{P} \setminus \mathcal{P}_i^\xi$ for all $i \in [s]$. We will now iterate over all mappings $\xi' : X \to C$ and for every such mapping we consider its induced partition of $\mathcal{P}_{-i}$. We denote the sets in this partition by $\mathcal{P}_1^{\xi',i}, \dots, \mathcal{P}_s^{\xi',i}$ and assume without loss of generality that $|\mathcal{P}_1^{\xi',i}| \leq \ldots \leq |\mathcal{P}_s^{\xi',i}|$. For each tuple $(V_{i_1}, \dots, V_{i_{s-1}}) \in \mathcal{P}_1^{\xi',i} \times \dots \times \mathcal{P}_{s-1}^{\xi',i}$ we add the set $\{V_{i_1}, \dots, V_{i_{s-1}}\}$ to $\mathcal{T}_i$.

Step 3. It remains to deal with configurations in $\Gamma \setminus \{\Phi^+\}$; by our assumption, we have $s(\Phi) \leq s-1$ for every $\Phi \in \Gamma \setminus \{\Phi^+\}$. We handle them in the same way as in Theorem 7.2, i.e., for each $i \in [s]$ and each $\Phi' \in \Gamma \setminus \{\Phi^+\}$ we add to $\mathcal{T}_i$ all sets of voters that correspond to occurrences of Φ' in E.

This completes the description of our reduction. The bound on its running time is immediate. Also, each $\mathcal{T}_i$, $i \in [s]$, only contains sets of size $s-1$ or less, i.e., we have constructed s instances of $(s-1)$-HITTING SET. Let A_i be an optimal solution for $(A, \mathcal{T}_i)$. To complete the proof, we will show that for each $i \in [s]$ (1) removing the voters in A_i from E results in an election that contains no configurations from Γ, and (2) if one can ensure that E contains no configurations from Γ by deleting a set of votes $\mathcal{P}' = \{V_{i_1}, \dots, V_{i_k}\}$, $k < \frac{n}{s-1}$, then $\mathcal{P}'$ is a hitting set for at least one of the instances $(A, \mathcal{T}_1), \dots, (A, \mathcal{T}_s)$.

To prove the first claim, fix $i \in [s]$ and consider a configuration $\Phi \in \Gamma$. If $\Phi \neq \Phi^+$, the claim is immediate: A_i intersects each set of voters corresponding to an occurrence of Φ in E, so by removing A_i we eliminate all occurrences of Φ. Now, suppose that $\Phi = \Phi^+ = \{\phi_1, \dots, \phi_s\}$. Consider some occurrence of Φ^+ in E; it corresponds to a mapping $\xi' : X \to C$ and a set of votes $\{V_{i_1}, \dots, V_{i_s}\}$, where $V_{i_j} \models_{\xi'} \phi_j$ for $j \in [s]$. If $\xi' = \xi$, then we have $\mathcal{P}_i^\xi \subseteq A_i$, and hence no vote in $(C, \mathcal{P} \setminus A_i)$ fulfills ϕ_i with respect to ξ'. Now, suppose that $\xi' \neq \xi$. If $V_{i_j} \in \mathcal{P}_i^\xi$ for some $j \in [s]$, then $V_{i_j} \in A_i$, and we are done. Otherwise we have $V_{i_j} \in \mathcal{P}_j^{\xi',i}$ for all $j \in [s]$. But then the set $\{V_{i_j} \mid 1 \leq j \leq s-1\}$ belongs to $\mathcal{T}_i$, and therefore A_i intersects it. This completes the proof of our first claim.

To prove the second claim, consider a set of votes $\mathcal{P}' = \{V_{i_1}, \dots, V_{i_k}\}$ such that $E' = (C, \mathcal{P} \setminus \mathcal{P}')$ contains no configurations from Γ. Note first that $\mathcal{P}'$ has to contain at least one of $\mathcal{P}_1^\xi, \dots, \mathcal{P}_s^\xi$; indeed, if $\mathcal{P} \setminus \mathcal{P}'$ intersects each of $\mathcal{P}_1^\xi, \dots, \mathcal{P}_s^\xi$, the votes in the intersection would correspond to an occurrence of Φ^+. Thus, suppose that $\mathcal{P}_i^\xi \subseteq \mathcal{P}'$ for some $i \in [s]$. We will now argue that $\mathcal{P}'$ is a hitting set for $(A, \mathcal{T}_i)$. Consider a set $S \in \mathcal{T}_i$. If S is a singleton that has been added to $\mathcal{T}_i$ during the first step, then we are done, since $S = \{V_j\}$ for some $V_j \in \mathcal{P}_i^\xi$, and $\mathcal{P}_i^\xi \subseteq \mathcal{P}'$. If S corresponds to an occurrence of a configuration in $\Gamma \setminus \{\Phi^+\}$, we are done, too, since $\mathcal{P}'$ hits all occurrences of this configuration.

Finally, suppose that $S = \{V_{j_1}, \dots, V_{j_{s-1}}\}$ where $V_{j_\ell} \in \mathcal{P}_\ell^{\xi',i}$ for all $\ell \in [s-1]$. We will now argue that if $\mathcal{P}' \cap S = \emptyset$, then $|\mathcal{P}'| \geq \frac{n}{s-1}$. To see this, suppose that $\mathcal{P}' \cap S = \emptyset$. Then $\mathcal{P}_s^{\xi',i} \subseteq \mathcal{P}'$. Indeed, if this is not the case, consider a vote $V_j \in \mathcal{P}_s^{\xi',i}$. All of the votes in $S \cup \{V_j\}$ are present in $E' = (C, \mathcal{P} \setminus \mathcal{P}')$, and hence E' contains an occurrence of Φ^+, a contradiction. As we also have $\mathcal{P}_i^\xi \subseteq \mathcal{P}'$ for some $i \in [s]$, and $\mathcal{P}_s^{\xi',i} \cap \mathcal{P}_i^\xi = \emptyset$, it follows that $|\mathcal{P}'| \geq |\mathcal{P}_s^{\xi',i}| + |\mathcal{P}_i^\xi|$. It remains to prove that $|\mathcal{P}_s^{\xi',i}| + |\mathcal{P}_i^\xi| \geq \frac{n}{s-1}$.

To establish this, let $y = |\mathcal{P}_i^\xi|$ and $z_j = |\mathcal{P}_j^{\xi',i}|$ for $j \in [s]$; we need to show that $y + z_s \geq \frac{n}{s-1}$. Recall that we have $z_1 \leq z_2 \leq \dots \leq z_s$, and hence $z_s \geq \frac{1}{s-1}(z_2 + \dots + z_s)$. Further, by our choice of ξ we have $y \geq z_1$ and therefore $z_2 + \dots + z_s = n - y - z_1 \geq n - 2y$. Thus, we obtain

$$y + z_s \geq y + \frac{n-2y}{s-1} \geq \frac{n}{s-1} + y\frac{s-3}{s-1} \geq \frac{n}{s-1},$$

where the last inequality follows since we assume $s \geq 3$.

We have argued that if $\mathcal{P}'$ fails to intersect some set in $\mathcal{T}_i$, then $|\mathcal{P}'| \geq \frac{n}{s-1}$. This completes the proof. □

The constraint on the true size of the optimal solution for Γ-VDEL in Theorem 7.3 may appear to be significant (and difficult to check). However, in Section 7.4 we will see that it does not affect our ability to design efficient approximation algorithms for Γ-VDEL.

Further, we remark that Theorem 7.3 only provides an improved reduction for Γ-VDEL, and not for Γ-CDEL. This is because there is no direct analog to the notion of a partitioning configuration for the latter problem.

7.3.1 Optimality of the Conversion

We will now show that the improved conversion is optimal by providing a reduction from d-HITTING SET to Γ-VDEL. In this reduction, any solution of a d-HITTING SET instance directly corresponds to a solution of the corresponding Γ-VDEL instance and, in particular, these two solutions have the same size. Consequently, improving the approximation factor for Γ-VDEL will lead to an improvement for d-HITTING SET. Since the approximation algorithm for d-HITTING SET is assumed to be optimal, we can also assume that our conversion algorithm is optimal. These consequences are discussed in more detail in Section 7.4. In this section, we only present the underlying reduction.

To make our reduction applicable to as many configuration-definable domain restrictions as possible, we introduce the notion of *solid subconfigurations*. The main intuition behind solid subconfigurations is that if a profile does not contain a solid subconfiguration on a set of candidates C_1 and neither on a set of candidates C_2, then it also does not contain this subconfiguration on $C_1 \cup C_2$. This property is essential for our reduction.

Definition 7.4. *Consider a configuration Φ with $X = X(\Phi)$, and a subset X' of X with $|X'| \geq 2$. Let $\Phi[X']$ be the restriction of Φ to X'. We say that $\Phi[X']$ is a* solid subconfiguration *if (i) $\Phi[X']$ is exact and (ii) for every $x, y \in X'$ there exists a condition $\phi \in \Phi$ where $x > y$ necessarily holds, i.e., ϕ implies $x > y$.*

Example 7.2. Consider the configuration Φ_α. The subconfiguration $\Phi_\alpha[\{a, b, c\}]$ is solid, but $\alpha[\{a, b, d\}]$ is not solid since neither ϕ_1 nor ϕ_2 implies $b > d$. The subconfiguration $\Phi_\alpha[\{b, d\}]$ is also not solid since it is not exact. $\dashv$

Theorem 7.4. *Consider a set of configurations Γ and a configuration $\Phi \in \Gamma$. Suppose that there exists an election $E = (C, \mathcal{P})$ with $\mathcal{P} = (V_1, \ldots, V_r)$ such that $r \geq 3$ and*

1. *E contains Φ;*

2. *for every $\Phi' \in \Gamma$ there exists a solid subconfiguration of Φ' such that for every $i \in [r-1]$ the election $(C, \mathcal{P} \setminus \{V_i\})$ does not contain this solid subconfiguration.*

Then there exist a polynomial-time reduction from $(r-1)$-HITTING SET to Γ-VDEL with the properties that

- *the $(r-1)$-HITTING SET instance has a size k solution if and only if the Γ-VDEL instance has a size k solution, and,*

- *the elements in the set A in the $(r-1)$-HITTING SET instance correspond one-to-one to voters in the Γ-VDEL instance.*

The consequence of this theorem is that solutions of Γ-VDEL instances can be directly translated to solutions of the $(r-1)$-HITTING SET instances. This will allow us to show that our improved conversion algorithm is (in some sense) optimal. First, let us prove this theorem.

Proof sketch. Let (A, S) be a $(r-1)$-HITTING SET instance given by $A = [u]$ and $\mathcal{T} = \{S_1, \ldots, S_q\}$. For each $j \in [q]$, we define a profile $\mathcal{R}^j = (W_1^j, W_2^j, \ldots, W_{u+k+1}^j)$. To define the votes W_i^j, we use the function $f(l, S)$ that returns the l-th smallest number in a set S. For example, $f(2, \{3, 4, 9\}) = 4$. We define W_i^j for $i \in [u+1, u+k+1]$ and $j \in [q]$, using the votes $V_1, \ldots, V_r$ from $\mathcal{P}$, as follows:

$$W_i^j = \begin{cases} V_l & \text{if } f(l, S_j) = i, \\ V_r & \text{otherwise.} \end{cases}$$

Note the following three facts. First, $f(l, S_j) \leq r-1$ since we have a $(r-1)$-HITTING SET instance. Second, for all $i \in [u+1, u+k+1]$, $W_i^j = V_r$ regardless of j. Third, the candidate set is identical for all profiles $\mathcal{R}^j$, $j \in [q]$.

Now, we would like to join these profiles together. For this, we assume distinct candidate sets $C_1, \dots, C_q$ for $\mathcal{R}^1, \dots, \mathcal{R}^q$, respectively. The joint profile is defined as follows:

$$\begin{aligned}\mathcal{R} = (&W_1^1 \succ W_1^2 \succ \cdots \succ W_1^q, \\ &W_2^1 \succ W_2^2 \succ \cdots \succ W_2^q, \\ &\vdots \\ &W_{u+k+1}^1 \succ W_{u+k+1}^2 \succ \cdots \succ W_{u+k+1}^q).\end{aligned}$$

This concludes our construction. We claim that it is possible to delete k votes from $\mathcal{R}$ to make it not contain the configurations from Γ if and only if $(A, \mathcal{T})$ has a size k hitting set.

($\leftarrow$) Let H be a hitting set of size $\leq k$. We remove the votes in $\mathcal{R}$ that correspond to H, i.e., for every $i \in H$ we remove the vote $W_i^1 \succ W_i^2 \succ \ldots \succ W_i^q$ from $\mathcal{R}$. Let $\mathcal{R}'$ be this reduced profile. Observe that none of the subprofiles $\mathcal{R}'[C_j]$, $j \in [q]$, contains a configuration in Γ; this is the case since by the theorem statement, for any $i \in [r-1]$, $(C, \mathcal{P} \setminus \{V_i\})$ does not contain a solid subconfiguration of every configuration in Γ and, consequently, $(C, \mathcal{P} \setminus \{V_i\})$ does not contain any configuration in Γ. It remains to verify that also $\mathcal{R}'$ does not contain a configuration in Γ. This is not possible since every solid subconfiguration would have to be contained in one of the subprofiles $\mathcal{R}'[C_j]$, $j \in [q]$ (due to the second condition in the theorem statement); we have already excluded this possibility.

($\rightarrow$) Let H be the set of vote indices that have been deleted. If for some set S_j, $j \in [q]$, $H \cap S_j = \emptyset$, then the profiles still contains a forbidden configuration regardless of the deletions. To see this, consider the votes in $\mathcal{R}[C_j]$ corresponding to indices contained in S_j ($r-1$ many) as well as some vote in $\{W_{r+1}^j, \dots, W_{u+k+1}^j\}$. (Note that it is not possible that $W_{r+1}, \dots, W_{u+k+1}$ are deleted with k deletions.) These votes contain a configuration Φ by the first condition of the theorem statement; this is a contradiction. Thus H is a hitting set. $\square$

This theorem holds for $r = 3$ and $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$, and, for $r = 4$ and Γ_C. The only set of configurations for which Theorem 7.4 does not hold is Γ_{sc}, since Φ_δ does not have any solid subconfigurations. This is because Φ_δ restricted to any proper subset of $X(\Phi_\delta)$ is no longer exact – and Φ_δ itself does not satisfy the second condition for being a solid subconfiguration. Theorem 7.4 not being applicable is reasonable since Γ_{sc}-VDEL is solvable in polynomial time and thus a reduction from HITTING SET would imply $\mathsf{P} = \mathsf{NP}$.

Let us exemplarily explain why this holds for $r = 3$ and Γ_{sp}. Consider the configuration Φ_α and the election $E = (C, \mathcal{P})$, where $C = \{a, b, c, d\}$, $\mathcal{P} = (V_1, V_2, V_3)$, $V_1 : dabc$, $V_2 : dcba$, $V_3 : dacb$. This election satisfies the conditions of Theorem 7.4. Indeed, it contains Φ_α in the first two votes. If one of these two votes is deleted, the resulting election no longer contains the solid subconfiguration $\Phi_\alpha[\{a, b, c\}]$ (see Example 7.2). Further, Φ_W is a solid subconfiguration by itself, and it can be eliminated by removing any of the three votes. Thus, 2-HITTING SET admits an approximation-preserving reduction to Γ_{sp}-VDEL.

Finally, let us remark that while the theorem works for $r = 4$ and Γ_C, it does not work for $r = 4$ and Γ_W, Γ_B, or Γ_M. The reason is that Γ_W, Γ_B, and Γ_M are partitioning and thus the second condition does not hold for $r = 4$. (In this case V_r would have to be a model of some of condition, say condition ϕ_i. Some vote, say vote V_i, has to be a model for ϕ_i. However, the election $(C, \mathcal{P} \setminus \{V_i\})$ now contains the configuration because V_r is a model for ϕ_i. This contradicts the second condition of Theorem 7.4.)

7.4 Approximation Algorithms

We now present the first application of our reductions. Since d-HITTING SET allows for a factor-d approximation [72], we are able to approximate Γ-VDEL and Γ-CDEL up to a constant factor for all sets of forbidden configurations Γ considered in Section 7.1.

Theorem 7.5. *Let Γ be a set of configurations and let $s = \max_{\Phi \in \Gamma} s(\Phi)$, $t = \max_{\Phi \in \Gamma} t(\Phi)$. Then Γ-VDEL admits a s-approximation algorithm, and Γ-CDEL admits a t-approximation algorithm. Moreover, if Γ contains a unique configuration Φ with $s(\Phi) = s$, and this configuration is partitioning, then Γ-VDEL admits an $(s-1)$-approximation algorithm.*

Proof. The first claim follows immediately from Theorem 7.2 and the fact that d-HITTING SET admits a polynomial-time d-approximation algorithm. Now, suppose that Γ contains a unique configuration Φ with $s(\Phi) = s$, and Φ is partitioning. We then use the reduction described in the proof of Theorem 7.3, and obtain s instances of $(s-1)$-HITTING SET. We run the $(s-1)$-approximation algorithm for $(s-1)$-HITTING SET, and obtain s sets $A_1, \dots, A_s$. We return the set of voters that corresponds to the smallest of these sets.

To see why this approach is correct, observe first that by Theorem 7.3 each of the sets $A_1, \dots, A_s$ corresponds to a feasible solution to our instance of Γ-VDEL. Now, let k be the size of the optimal solution for our instance of Γ-VDEL. If $k < \frac{n}{s-1}$, then by Theorem 7.3 one of our instances of $(s-1)$-HITTING SET has a hitting set of size k, so $\min_{i \in [s]} |A_i| \leq (s-1)k$. Otherwise we have $(s-1)k \geq n$, so even the solution that deletes all voters (and hence any of the sets A_i) is within a factor of $(s-1)$ from optimal. □

Corollary 7.6. *For $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$, the problem Γ-VDEL can be approximated within a factor of 2, and Γ_C-VDEL can be approximated within a factor of 3. Moreover, the problem Γ-CDEL can be approximated within a factor of 3 for $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_C\}$, within a factor of 4 for $\Gamma = \Gamma_{gs}$, and within a factor of 6 for $\Gamma = \Gamma_{sc}$.*

Corollary 7.7. *Assuming the Unique Games Conjecture, the approximation results for Γ-VDEL with $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_C, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$ are optimal.*

Proof. The d-approximation of d-HITTING SET is optimal under the assumption that the Unique Games Conjecture holds [103]. As remarked at the end of Section 7.3.1, Theorem 7.4 holds for $r = 3$ and $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$ and for $r = 4$ and Γ_C. Thus, a 3-approximation is optimal for Γ_C and a 2-approximation for the other domain restrictions – which are exactly the approximation ratios we obtained. □

d	2	3	4	5	6
c_d	1.28	2.08	3.15	4.11	5.07

Table 7.1: Currently best algorithms for d-HITTING SET with a runtime of $\mathcal{O}^*({c_d}^k)$, where k is the optimal solution size

7.5 Fixed-Parameter Algorithms

Fixed-parameter algorithms for Γ-VDEL can be obtained by utilizing fpt algorithms for d-HITTING SET [51, 77, 137]. The currently best runtimes for d-HITTING SET are displayed in Table 7.1.

Theorem 7.8. *Let Γ be a set of configurations, and let $s = \max_{\Phi\in\Gamma} s(\Phi)$, $t = \max_{\Phi\in\Gamma} t(\Phi)$. Then Γ-VDEL can be solved in time $\mathcal{O}^*({c_s}^k)$, and Γ-CDEL can be solved in time $\mathcal{O}^*({c_t}^k)$, where k is the size of the optimal solution and c_s is taken from Table 7.1. Moreover, if $k < n/2$, Γ contains a unique configuration Φ with $s(\Phi) = s$, and Φ is partitioning, then Γ-VDEL can be solved in time $\mathcal{O}^*(c_{s-1}^k)$, where k is the size of the optimal solution.*

Corollary 7.9. *For $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$, Γ-VDEL can be solved in time $\mathcal{O}^*(1.28^k)$ if $k < n/2$ and in time $\mathcal{O}^*(2.08^k)$ otherwise; Γ_C-VDEL can be solved in time $\mathcal{O}^*(2.08^k)$.*

For $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_C\}$, Γ-CDEL can be solved in time $\mathcal{O}^*(2.08^k)$, Γ_{gs}-CDEL in time $\mathcal{O}^*(3.15^k)$ and Γ_{sc}-CDEL in time $\mathcal{O}^*(5.07^k)$.

The results in Corollary 7.9 for $k < n/2$ can be considered optimal in the following sense: Observe that Theorem 7.4 shows that for instances with $k < n/2$ there is a reduction from VERTEX COVER, i.e., 2-HITTING SET, to VDEL. This reduction is also an fpt reduction. Thus, any fpt runtime improvement on VDEL for $k < n/2$ would also imply an improvement for VERTEX COVER.

7.6 Deleting Almost All Votes

The approximation algorithm described in Section 7.4 is useful when the size of the optimal solution for Γ-VDEL does not exceed $n/2$. However, it may also be the case that, to eliminate configurations in Γ, we need to delete almost all voters. In this case, it is trivial to find a 2-approximate solution to Γ-VDEL: simply deleting all voters provides a 2-approximation. Thus, a more fine-grained approach is to try to approximate the number of *surviving* voters; we refer to this variant of our problem as Γ-VDEL$^-$.

Γ-VDEL $^-$	
Instance:	An election $E = (C, \mathcal{P})$.
Question:	Find the largest k such that for some $\mathcal{P}'$ with $\lvert\mathcal{P}'\rvert = k$ the election $(C, \mathcal{P}')$ contains no configurations from Γ.

It turns out that Γ-VDEL$^-$ is hard to approximate for many sets of configurations Γ.

Theorem 7.10. *Consider a set of configurations* Γ *and a configuration* $\Phi \in \Gamma$. *Suppose that there exists an election* $E = (C, \mathcal{P})$ *with* $\mathcal{P} = (V_1, \ldots, V_r)$ *such that* $r \geq 3$ *and*

1. E *contains* Φ*;*
2. *for every* $\Phi' \in \Gamma$ *there exists a solid subconfiguration of* Φ' *such that for every* $i \in [r-1]$ *the election* $(C, \mathcal{P} \setminus \{V_i\})$ *does not contain this solid subconfiguration.*

Then Γ-VDEL$^-$ *cannot be approximated by a constant factor unless* $\mathsf{P} = \mathsf{NP}$.

Proof sketch. We reduce from INDEPENDENT SET, which cannot be approximated by a constant factor (not even within $n^{1-\epsilon}$) unless $\mathsf{P} = \mathsf{NP}$ [95, 144]. The INDEPENDENT SET problem asks whether, given a graph $G = (N, T)$, there exists a set of vertices $S \subseteq N$ with $|S| \leq k$ such that there are no edge in T between vertices in S. Let $N = \{x_1, \ldots, x_{|N|}\}$ and $T = \{e_1, \ldots, e_{|T|}\}$. We construct a Γ-VDEL instance, similar to the construction in the proof of Theorem 7.4. Let $u = |N| + (k+1) \cdot (r-2)$. For each $j \in \{1, \ldots, |T|\}$, we define a profile $\mathcal{R}_j = (W_1^j, \ldots, W_u^j)$. We join these subprofiles to the profile

$$\mathcal{R} = (W_1^1 \succ W_1^2 \succ \ldots \succ W_1^{|T|}, \ldots, W_u^1 \succ W_u^2 \succ \ldots \succ W_u^{|T|})$$

(again by making the candidate sets distinct, as in the proof of Theorem 7.4). Let us define the votes W_i^j, $i \in \{1, \ldots, u\}$ and $j \in \{1, \ldots, |T|\}$:

$$W_i^j = \begin{cases} V_1 & \text{if } e_j = \{x_i, x_{i'}\} \text{ with } i < i', \\ V_2 & \text{if } e_j = \{x_i, x_{i'}\} \text{ with } i' < i, \\ V_r & \text{if } x_i \notin e_j, \\ V_l & \text{if } |N| + (l-3) \cdot (k+1) < i \leq u. \end{cases}$$

Note that, regardless of j,

$$W_{|N|+1}^j = W_{|N|+2}^j = \cdots = W_{|N|+k+1}^j = V_3,$$

$$W_{|N|+k+2}^j = W_{|N|+k+3}^j = \cdots = W_{|N|+2k+2}^j = V_4, \ldots$$

$$\ldots, W_{|N|+(k+1)(r-3)+1}^j = W_{|N|+(k+1)(r-3)+2}^j = \cdots = W_{|N|+(k+1)(r-2)}^j = V_r.$$

The intuition here is that for each edge $x_i, x_{i'}$ ($i < i'$) either the vote $W_i^1 \succ W_i^2 \succ \ldots \succ W_i^{|T|}$ (containing an occurrence of V_1) or the vote $W_{i'}^1 \succ W_{i'}^2 \succ \ldots \succ W_{i'}^{|T|}$ (containing an occurrence of V_2) has to be deleted since deleting all occurrences of either $V_3, V_4, \ldots, V_r$ is not possible. We claim that deleting k votes from $\mathcal{R}$ to make it avoid every configuration in Γ is possible if and only if there is a size k independent set. The proof is similar to the one of Theorem 7.4. □

Corollary 7.11. *For* $\Gamma \in \{\Gamma_W, \Gamma_B, \Gamma_M, \Gamma_C, \Gamma_{sp}, \Gamma_{scv}, \Gamma_{gs}\}$, Γ-VDEL$^-$ *cannot be approximated by a constant factor unless* $\mathsf{P} = \mathsf{NP}$.

Γ	VDEL	CDEL
Single-peaked / Single-caved	2	P
Single-crossing	P	6
Best-/Medium-/Worst-restricted	2	3
Value-restricted	3	3
Group-separable	2	4

Table 7.2: Approximation algorithms for Γ-VDEL and Γ-CDEL

We can also characterize the parameterized complexity of Γ-VDEL$^-$. Here, we require the decision problem corresponding to Γ-VDEL$^-$, i.e., given $k > 0$, is there a subset of at least k voters such that the corresponding profile contains no configuration from Γ?

Theorem 7.12. Γ-VDEL$^-$ *parameterized by the size of an optimal solution, i.e., the number of surviving voters, is* W[1]*-complete.*

Proof sketch. First, observe that the reduction in the proof of Theorem 7.10 is also an fpt reduction. Since INDEPENDENT SET is W[1]-hard, we obtain W[1]-hardness from Theorem 7.10.

W[1]-membership can be shown by encoding a VDEL instance in a model checking problem of a propositional Σ_1 formula. For a formal definition of this model checking problem we refer the reader to the book by Flum and Grohe [80]. □

7.7 Summary

We have investigated the complexity of approximating the distance between a given election and a restricted preference domain, for two natural distance measures and many well-known restricted preference domains. Our results are broadly positive: they include polynomial-time approximation algorithms whose approximation ratio is bounded by a small constant (summarized in Table 7.2) and reasonably fast fpt algorithms (summarized in Table 7.3). However, for the variant of the voter deletion problem where the goal is to approximate the number of surviving voters, our results are rather negative.

The reader may wonder if improving the approximation ratio of our algorithms, e.g., for Γ_{sp}-VDEL from 3 to 2, by going through a more complicated reduction was worth the effort. Observe, however, that the runtime of the algorithms for nearly single-peaked elections typically grows exponentially (or faster) with the distance from the single-peaked domain [75]; thus, a constant-factor improvement in approximation ratios translates into significant improvement in the running time of these algorithms.

Γ	FPT runtime for VDEL		FPT runtime for CDEL
	$k < n/2$	$k \geq n/2$	
Single-peaked / Single-caved	$\mathcal{O}^*(1.28^k)$	$\mathcal{O}^*(2.08^k)$	P
Single-crossing	P	P	$\mathcal{O}^*(5.07^k)$
Best-/Medium-/Worst-restricted	$\mathcal{O}^*(1.28^k)$	$\mathcal{O}^*(2.08^k)$	$\mathcal{O}^*(2.08^k)$
Value-restricted	$\mathcal{O}^*(2.08^k)$	$\mathcal{O}^*(2.08^k)$	$\mathcal{O}^*(2.08^k)$
Group-separable	$\mathcal{O}^*(1.28^k)$	$\mathcal{O}^*(2.08^k)$	$\mathcal{O}^*(3.15^k)$

Table 7.3: Fpt algorithms for Γ-VDEL and Γ-CDEL

CHAPTER 8

Structure in Incomplete Preferences

This chapter is based on the publication *Incomplete preferences in single-peaked electorates* [111].

Both human and automated decision making often have to rely on incomplete information. The same issue arises in joint decision making – voting – in multi-agent systems. For example, the majority of preference data collected on `preflib.org` [117] is incomplete. Konczak and Lang [71] distinguish two main sources of incompleteness: The first one is *intrinsic* incompleteness where the voter is unable[1] or unwilling to give complete information, i.e., a total order on all candidates. The second one is *epistemic* incompleteness where the voters do have preferences specified by total orders but at the time of decision making these total orders are not fully available. Also a combination of these two scenarios is possible.

Whereas complete preferences are usually modeled as total orders, incomplete preferences can be modeled as partial orders and are therefore a more general concept. In particular, the determination of winners becomes harder since voting protocols usually require total orders. It is therefore necessary to consider *completions* of incomplete votes. Completions of incomplete votes are total orders that are compatible with the original partial orders. The determination of possible and necessary winners in incomplete elections is often intractable and thus a fast winner determination is not feasible [26, 27, 71, 124, 138, 142].

A popular approach to deal with hardness of voting problems is to consider domain restrictions. The most common restriction is *single-peakedness* [30] (see Section 2.2 for a definition). For example, computing the winner of a Dodgson or Kemeny election, though Θ_2^P-complete in general [96, 97], can be done in polynomial time for single-peaked elections [41]. Also the complexity of manipulation and control problems often decreases [76]. These results let us hope that efficient, polynomial time algorithms for computing possible and necessary winners of single-peaked incomplete elections could be found. Walsh [138] started investigating this issue and also

[1]In the case that the voter is unable to provide complete information because, for example, two candidates are equally preferred by the voter, the term "incomplete" is not really accurate; a partial order might constitute complete information in such a case. For our studies, however, this subtlety is not relevant.

pointed out a central question in that regard: What happens if the axis for which the incomplete preference profile is single-peaked is not given as part of the input but has to be determined?

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

• We prove that determining whether an incomplete preference profile is single-peaked is NP-complete. This is in contrast to the case of complete preferences for which single-peakedness can be determined in linear time [71]. Furthermore, we strengthen this result by showing that NP-completeness still holds if one voter completely specifies his preferences.

Apart from these hardness results, this chapter contains four polynomial time algorithms:

• The first algorithm requires that the preference profile must contain at least one complete vote, i.e., a total order. The algorithm is applicable to weak orders (see Figure 8.1 for an example and Section 8.1 for a definition). We obtain a runtime of $\mathcal{O}(m \cdot n)$. This algorithm is an improvement over the algorithm by Escoffier, Lang and Ötztürk [71] since it is applicable to a broader class of preference profiles (weak orders instead of total orders) while maintaining its runtime.

• Our second algorithm is 2-SAT based. It also requires a total order but is applicable to local weak orders, which are a generalization of weak orders. This more general algorithm does not run in linear time but requires $\mathcal{O}(m^3 \cdot n)$ time.

• In contrast to the previous two algorithms, the third algorithm does not require the profile to contain a total order. However, it is restricted to top orders. Top orders rank an arbitrary number of top candidates; all remaining candidates are ranked last and incomparable to one another (see Figure 8.1 for an example). This algorithm has a runtime of $\mathcal{O}(m^2 \cdot n)$.

• Finally, we consider the problem of determining single-peakedness for an already given axis. We prove this problem to be polynomial-time solvable even for incomplete profiles consisting of partial orders.

8.1 Incomplete Preferences

In this chapter, preferences are represented by different types of orders (see Figure 8.1 for examples). The most general type are partial orders. A *partial order* P on a set X is a reflexive, antisymmetric and transitive binary relation on X. We say that *y is ranked above x* if xPy holds. If for two elements $x, y \in X$ neither xPy nor yPx holds, these two elements are *incomparable*. A partial order where the incomparability relation is transitive is called a *weak order*. A weak order can thus be considered a total order with ties. Weak orders are also referred to as bucket orders (elements that tie are in the same "bucket"), cf. [73]. A weak order where all incomparable elements are minimal is called *top order*. The *ranked* candidates of a top order T are those that are not incomparable to any other candidate. We would like to remark that top orders appear as *top lists* in [65, 74] and as top-truncated votes in [24]. A partial order with no incomparable elements is called *total order*. Any partial order P can be *extended* to some total order T such that aPb implies aTb; T is then a (not necessarily unique) *extension* of P. Finally, we define a *local weak order* P on a set X to be a partial order on X with the following property: there exist sets X_1, X_2 with $X_1 \cup X_2 = X$ such that the elements in X_1 are incomparable to all other elements in X and the profile P restricted to X_2 is a weak order. Intuitively, a local weak

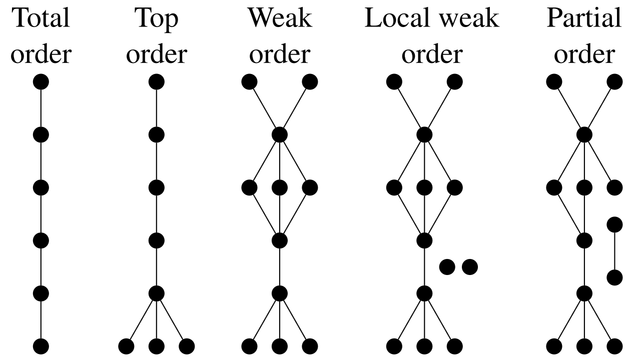


Figure 8.1: The order zoo: examples of different types of orders that are used to specify preferences

order is a weak order together with some isolated elements for which absolutely no information is available. Note that we do not distinguish between tied and incomparable elements in this paper; both are treated in the same way.

In this chapter, total orders are denoted by $\langle c_1 > c_2 > \ldots > c_k \rangle$; the brackets allow us to unambiguously denote total orders consisting of one or even zero elements, i.e, we use $\langle\rangle$ to denote the empty order relation. For top orders, we write $\langle c_1 > c_2 > \ldots > c_k > \bullet \rangle$ to denote a top order where $c_1, \ldots, c_k$ are ranked as stated and all other elements (usually the remaining candidates in C) are ranked last, i.e., are minimal elements. We sometimes use set operators ($\cup, \cap, \setminus$) on top orders with the intended meaning that we apply these operators to the corresponding sets of ranked candidates.

We would now like to address the usefulness of these types of orders for expressing preferences. Total orders allow the voter to fully specify a ranking of options. Given a large set of options, this might be unfeasible. Partial orders, on the other hand, allow the voter to specify the relative order of any pair of options. Thus they can be seen as a very general formalism for representing incomplete preferences. They are compatible with total orders in the sense that partial orders can always be extended to total orders. Weak orders are less general than partial orders but arise in many natural scenarios. For example every real-valued utility function implies a weak order (candidates with the same utility tie, i.e., are incomparable). Local weak orders correspond to partial real-valued utility functions and thus arise in scenarios where voters do not have knowledge about all candidates. If the elicitation of preferences is costly, one might ask only for the most important (top ranked) options of each voter. In such a case, top orders arise. Top orders also are the natural type of order for specifying preferences in some scoring protocols. We will further comment on scoring protocols and top orders at the end of the chapter.

Votes are considered to be either partial, local weak, weak, top or total orders. A tuple $(V_1, \ldots, V_n)$ of votes is called a *(preference) profile of* $\{$*partial orders, weak orders, top orders, total orders*$\}$, depending on the type of orders.

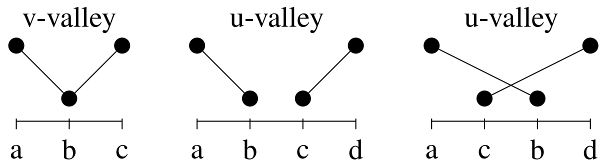


Figure 8.2: What v-valleys and u-valleys may look like

8.2 Single-peaked Profiles

We start by repeating the definition of single-peaked profiles of total orders and then extend this definition to partial orders. Our definition of single-peakedness for profiles of total orders (Definition 2.2) as well as for profiles of partial orders is based on so-called *valleys*. Here, we distinguish two types of valleys. The first type, v-valleys, is the one already used in Definition 2.2.

Definition 8.1 (v-valleys)**.** *Let V be a partial order on C. The vote V* contains a v-valley with respect to an axis A *if there exist $c_1, c_2, c_3 \in C$ such that $c_1 < c_2 < c_3$, $c_2 \prec c_1$ and $c_2 \prec c_3$.*

Recall that a profile $\mathcal{P}$ of total orders is single-peaked with respect to A if no vote $V \in \mathcal{P}$ contains a v-valley with respect to A (and thus every vote has only a single "peak").

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

Definition 8.2. *Let $\mathcal{P} = (V_1, \ldots, V_n)$ be a profile of partial orders. The profile $\mathcal{P}$ is single-peaked with respect to an axis A if for every $k \in \{1, \ldots, n\}$, V_k can be extended to a total order V_k' such that the profile of total orders $\mathcal{P}' = (V_1', \ldots, V_n')$ is single-peaked with respect to A.*

While it is also conceivable to require that every extension is single-peaked, this would yield an extremely restrictive definition. In this sense, our definition seems to be preferable.Next, we want to find an equivalent definition based on valleys, for which we also require u-valleys:

Definition 8.3 (u-valleys)**.** *Let V be a partial order on C. The vote V* contains a u-valley with respect to A *if there exist distinct elements $a, b, c, d \in C$ with $a < b < d$ and $a \succ b$ as well as $a < c < d$ and $d \succ c$.*

In Figure 8.2 a graphical representation of v- and u-valleys is shown. These two types of valleys allow a characterization of single-peakedness for profiles of partial orders.

Lemma 8.1. *Let $\mathcal{P} = (V_1, \ldots, V_n)$ be a profile of partial orders. The following two statements are equivalent.*

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

(ii) Every vote $V \in \mathcal{P}$ contains neither a u-valley nor v-valley with respect to A.

Proof. Statement (i) implies Statement (ii) since if some V_k contained a u-valley in the sense of Definition 8.3 then every extension V'_k would contain a v-valley. More concretely, if $a, b, c, d \in C$ form a u-valley in V_k then any extension of V_k either contains a v-valley with respect to A on the candidates a, b, c or on b, c, d. Furthermore, if V_k contained a v-valley then so would every extension.

For the other direction, we show that a vote V not containing a valley can be extended to a total order that is single-peaked with respect to A. We are going to recursively define its extension V' starting with the last ranked candidate. Let $V'(1)$ denote the last ranked candidate, $V'(2)$ the second-to-last, etc. For the definition we require two functions:

- $\mathit{min}_A(X)$ is the smallest (leftmost) candidate in X with respect to the axis A.
- $\mathit{max}_A(X)$ is the largest (rightmost) candidate in X with respect to the axis A.

We now define for every $i \in \{1, \ldots, m\}$,

$$X_i = C \setminus \{V'(1), \ldots, V'(i-1)\} \text{ and}$$
$$V'(i) = \begin{cases} \mathit{max}_A\,(X_i) & \text{if } \mathit{max}_A\,(X_i) \text{ is } \succ\text{-minimal in } V[X_i] \\ \mathit{min}_A\,(X_i) & \text{otherwise.} \end{cases}.$$

This definition immediately yields that V' is single-peaked with respect to A: By always choosing one of the two outermost candidates on A (that have not yet been chosen) for the next higher ranked candidate, valleys cannot arise.

It remains to show that V' is indeed an extension of V, i.e., we have to show that for every pair of candidates $a, b \in C$, $a \succ b$ implies $a \succ' b$. Towards a contradiction assume that $a \succ b$ and $b \succ' a$. Let $i \in \{1, \ldots, m\}$ such that $V'(i) = a$. We have to consider two cases: $a = \mathit{min}_A(X_i)$ and $a = \mathit{max}_A(X_i)$.

Let $a = \mathit{min}_A(X_i)$ and $d = \mathit{max}_A(X_i)$. Since a has been chosen as $V'(i)$, we know that there has to exist a $c \in X_i$ with $d \succ c$. Observe that $a < c < d$ has to hold. Furthermore, either $a < b < d$ or $b = d$ holds. If $a < b < d$ holds then a, b, c, d for a u-valley. If $b = d$ then $a < c < b$, $a \succ b$ and $b \succ c$ holds: a v-valley. Both cases contradict our assumption that V does not contain a valley with respect to A.

Now, let $a = \mathit{max}_A(X_i)$. This immediately yields a contradiction since $a \succ b$ and a is not a $\succ$-minimal element. $\square$

This lemma immediately yields a polynomial time algorithm for checking whether an incomplete profile is single-peaked with respect to a given axis:

Theorem 8.2. *Checking whether a profile of partial orders is single-peaked with respect to a given axis can be done in $\mathcal{O}(n \cdot m^4)$ time.*

Proof. For every quadruple of candidates and every vote, one has to check whether a u- or v-valley arises. $\square$

Let $\mathcal{T} \in$ {partial order, local weak order, weak order, top order, total order} be a type of order. In this chapter we are going to study the following problem:

$\mathcal{T}$ SINGLE-PEAKED CONSISTENCY *Instance:* A profile $\mathcal{P}$ of type $\mathcal{T}$ and a set of candidates C. *Question:* Is $\mathcal{P}$ single-peaked consistent?

Note that in contrast to Theorem 8.2, the input of this problem does not include an axis. The TOTAL ORDER SINGLE-PEAKED CONSISTENCY problem is known to be solvable in polynomial time [23,61,71]. In the next section, we show that this is likely not to be the case for partial orders and even local weak orders.

8.3 Hardness Results

Theorem 8.3. *The* LOCAL WEAK ORDER SINGLE-PEAKED CONSISTENCY *problem is* NP-*complete.*

Proof. We reduce from the NP-complete BETWEENNESS problem [121]. A BETWEENNESS instance consists of a finite set S and a set T containing (ordered) triples of distinct elements of S. The decision problem asks whether there is a total order L such that for every triple $(a, b, c) \in T$ we have either $aLbLc$ or $cLbLa$. Intuitively, a triple $(a, b, c) \in T$ corresponds to the constraint that b has to lie "in between" a and c on the total order L.

We construct an incomplete election $(C, \mathcal{P})$ with $C = S$, i.e., we identify elements in S with candidates. The preference profile $\mathcal{P}$ consists of two votes for each triple (a, b, c): the partial order $\{a \succ c, b \succ c\}$ and the partial order $\{b \succ a, c \succ a\}$. These two votes form a valley on any axis with c between a and b and on any axis with a between b and c. Thus b has to be between a and c on any single-peaked axis. We are now going to show that $\mathcal{P}$ has a single-peaked extension profile if and only if the BETWEENNESS instance is a Yes-instance.

" $\Rightarrow$" Assume that there exists an extension profile $\mathcal{P}^{ext}$ of $\mathcal{P}$ and an axis A such that $\mathcal{P}^{ext}$ is single-peaked with respect to A. By Lemma 8.1 we know that this implies that no v-valleys exist. Since for every triple $(a, b, c) \in T$ both the vote $\{a \succ c, b \succ c\}$ and $\{b \succ a, c \succ a\}$ are contained in $\mathcal{P}$, we have that neither $a < c < b$, $b < c < a$, $b < a < c$ nor $c < a < b$ can hold. Consequently it has to hold that either $a < b < c$ or $c < b < a$ holds and thus b is "in between" a and c.

" $\Leftarrow$" Assume that there exists a set T such that all constraints in T are satisfied. It is easy to verify that $(C, \mathcal{P})$ is single-peaked with respect to T. $\square$

Corollary 8.4. PARTIAL ORDER SINGLE-PEAKED CONSISTENCY *is* NP-*complete.*

The proof of Theorem 8.3 uses elections where the votes contain very little information: only two pairs of candidates are comparable in each vote. We know that determining single-peaked consistency is possible in polynomial time if every vote is a total order, i.e., all votes contain complete information. Now the question arises: what happens if only a single voter provides complete information? Having a single completely specified vote has been found to be helpful in a related context: it allows for the efficient elicitation of single-peaked preferences using only few comparison queries [52] and thus the communication complexity of preference elicitation is reduced. However, in our case such a voter does not provide enough additional information for a decrease in (computational) complexity.

Theorem 8.5. *The* PARTIAL ORDER SINGLE-PEAKED CONSISTENCY *problem is* NP*-complete even if the preference profile contains a total order.*

Proof. We reduce from SET SPLITTING: Let X be a finite set. Given a collection Z of subsets of X, is there a partition of X into two subsets X_1 and X_2 such that no subset of Z is contained entirely in either X_1 or X_2? This problem is NP-complete even if all sets in Z have cardinality three [86].

Let $X = \{c_1, \ldots, c_m\}$. For the construction, we identify the elements of X with candidates and add an additional candidate x. For each set $\{c_i, c_j, c_k\} \in Z$ with $i < j < k$ we introduce one vote: $\{c_i \succ c_j, x \succ c_k\}$. In addition, we add the vote $x \succ c_m \succ \cdots \succ c_1$. We claim that the resulting preference profile $\mathcal{P}$ is single-peaked if and only if (X, Z) is a SET SPLITTING yes-instance.

Assume that $\mathcal{P}$ is single-peaked and let A be the corresponding axis. We define X_1 to be the candidates on A left of x and X_2 those that right of x. We will show that there is no subset of Z entirely contained in X_1 or X_2. Towards a contradiction assume that $\{c_i, c_j, c_k\} \in Z$ with $i < j < k$ are contained in X_1. Then it has to hold that, on A, c_i, c_j, c_k are all left of x. Furthermore, from the vote $x \succ c_m \succ \cdots \succ c_1$ then it follows that the relative order on A of x, c_i, c_j, c_k has to be $c_i < c_j < c_k < x$. However this order is not single-peaked for the vote $\{c_i \succ c_j, x \succ c_k\}$. Assuming that $\{c_i, c_j, c_k\} \in Z$ are contained in X_2 leads to the same contradiction. Thus, X_1 and X_2 indeed certify that (X, Z) is a yes-instance.

For the other direction, assume that (X, Z) is a yes-instance and X_1 and X_2 the partition. Let an axis A be defined as the elements in X_1 with indices in increasing order followed by x followed by the elements in X_2 with indices in decreasing order. We claim that A is an axis for $\mathcal{P}$. Clearly, the vote $x \succ c_m \succ \cdots \succ c_1$ is single-peaked with respect to A. Let us consider a vote $\{c_i \succ c_j, x \succ c_k\}$ with $i < j < k$. Since X_1 and X_2 are a valid partition, at least one of c_i, c_j, c_k has to be left of x and another one right. This rules out that a u-valley is formed and thus all votes are single-peaked with respect to A. $\square$

It is important to note that – in contrast to the NP-hardness result in Theorem 8.3 – in this proof we make use of u-valleys instead of v-valleys. This means in particular that this hardness result does not hold for weak orders, which cannot contain u-valleys. This is not incidental: in the next section, we present a polynomial-time algorithm for weak orders.

8.4 The Guided Algorithm

In this section, we present a polynomial time algorithm for profiles of weak orders. This algorithm requires that the profile contains at least one total order to guide the placement of candidates on the axis. We call this vote the *guiding vote*. Clearly, not all profiles of partial orders possess a guiding vote. In particular, the profiles constructed in the proof of Theorem 8.3 do not possess one.

Theorem 8.6. *If the profile contains a total order, the* WEAK ORDER SINGLE-PEAKED CONSISTENCY *problem can be solved in $\mathcal{O}(m \cdot n)$ time.*

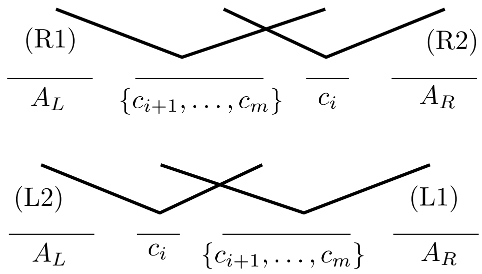


Figure 8.3: Graphical representation of the conditions testing whether c_i can be placed on the right-hand side ((R1), (R2)) or on the left-hand side ((L1), (L2))

We will refer to Algorithm 5, which Theorem 8.6 is based on, as the *Guided Algorithm*. Without loss of generality, we assume that the guiding vote is $c_m \succ c_{m-1} \succ \cdots \succ c_1$, i.e., we number the candidates based on the guiding vote.

The most important observation for detecting single-peakedness in weak orders is that only v-valleys can arise. An u-valley would violate the condition that the incomparability relation is transitive.

The algorithm has a simple structure: The lowest ranked candidate in the guiding vote, c_1, is placed on the rightmost position of the axis (The leftmost position would work as well.) Starting with the second lowest ranked candidate, c_2, in the guiding vote, the candidates are successively placed on the axis – either at the leftmost or rightmost still available position. The lists A_L and A_R correspond to the left-hand and right-hand side of the axis. For each candidate, we test whether it can be placed on the right-hand side or left-hand side without creating a valley. If only one of these options is viable, the candidate is placed accordingly. If both left and right are possible, we place the candidate arbitrarily right. If neither is possible, the preference profile is not single-peaked.

Testing whether a vote V_k imposes restrictions on the placement of a candidate is achieved by four conditions. These conditions distinguish four categories of candidates: candidates in A_R, candidates in A_L, candidates that have not yet been placed ($C_{>i} = \{c_{i+1}, \ldots, c_m\}$) and the candidate that is currently under consideration (c_i). We are only checking for valleys that include c_i. This gives rise to the following four conditions: (R1) and (R2) test whether placing c_i on the right-hand side leads to valleys, (L1) and (L2) do the same for the left-hand side. Figure 8.3 displays a graphical representation. Note that it is not necessary to verify whether a v-valley arises with $a_l \succ c_i$ and $a_r \succ c_i$, where $a_l \in A_L$ and $a_r \in A_R$; such a valley would have already be detected at an earlier stage of the algorithm. Since we only consider weak orders, we do not have to consider every candidate triple possibly fulfilling these conditions but have to check only maximal or minimal candidates. More specifically, checking whether there is a candidate $c \in A_L$ and $c' \in C_{>i}$ with $c \succ c'$ is equivalent to whether any maximal element in A_L is preferred to some minimal element in $C_{>i}$. For $k \in [n]$, let $min_k(X)$ denote a function that picks some arbitrary element in X that is minimal with respect to $\succ_k$. The function $max_k(X)$ is defined analogously. Now, we can formally define the four conditions:

$$c_i \succ_k \mathit{min}_k(C_{>i}) \text{ and } \mathit{max}_k(A_L) \succ_k \mathit{min}_k(C_{>i}) \tag{R1}$$

$$\mathit{max}_k(C_{>i}) \succ_k c_i \text{ and } \mathit{max}_k(A_R) \succ_k c_i \tag{R2}$$

$$c_i \succ_k \mathit{min}_k(C_{>i}) \text{ and } \mathit{max}_k(A_R) \succ_k \mathit{min}_k(C_{>i}) \tag{L1}$$

$$\mathit{max}_k(C_{>i}) \succ_k c_i \text{ and } \mathit{max}_k(A_L) \succ_k c_i \tag{L2}$$

Using these four definitions, we can give a succinct description of the algorithm (Algorithm 5).

Algorithm 5: The Guided Algorithm

Input: A set of candidates C, a preference profile of weak orders $\mathcal{P} = (V_1, \dots V_n)$ including a guiding vote $c_m \succ c_{m-1} \succ \dots \succ c_1$.
Output: An axis A or *not_single_peaked*.

$A_L \leftarrow \langle\rangle$
$A_R \leftarrow \langle c_1 \rangle$
for $i \leftarrow 2 \dots m$ **do**
    right $\leftarrow$ *true*; *left* $\leftarrow$ *true*
    for $k \leftarrow 2 \dots n$ **do**
        if *Condition* (R1) *or* (R2) *holds* **then**
            right $\leftarrow$ *false*
        if *Condition* (L1) *or* (L2) *holds* **then**
            left $\leftarrow$ *false*
    if *right* $=$ *true* **then**
        $A_R \leftarrow \langle c_i < A_R \rangle$
    else
        if *left* $=$ *true* **then**
            $A_L \leftarrow \langle A_L < c_i \rangle$
        else
            return `not_single_peaked`
return $A_L < A_R$

Theorem 8.6 claims that the Guided Algorithm requires $\mathcal{O}(m \cdot n)$ time. This is only possible if the conditions can be checked in constant time. Thus, the minima and maxima have to be computable in constant time. For $\mathit{max}_k(A_L)$ and $\mathit{max}_k(A_R)$ this is easily possible by storing and updating these two values. If c_i is placed left, we update $\mathit{max}_k(A_L)$ in case c_i is the new maximum (with respect to $\succ_k$); if c_i is placed right, we proceed analogously $\mathit{max}_k(A_R)$. For computing a minimal value of $C_{>i}$, observe that the set $C_{>i}$ becomes smaller with increasing i. Thus, a minimal value of $C_{>i}$ might disappear at some point and a new (larger) value has to be chosen. The new minimum is the smallest element (with respect to $\succ_k$) in $C_{>i}$ that is at least as large as the old minimum. If we maintain pointers to the minimum elements, the amortized cost of this update procedure is $\mathcal{O}(1)$. A maximal value of $C_{>i}$ can be found analogously.

We are going to show that the Guided Algorithm is correct. For this, we require the following definition and the two following lemmas.

Definition 8.4. *In a weak order, we write $a \succsim b$ to denote that either $a \succ b$ holds or a is incomparable to b.*

Lemma 8.7. *We consider the Unguided Algorithm at any given point during its runtime. In particular, we consider the sets A_L, A_R and $\{c_i, \dots, c_m\}$. Let $V_k \in \mathcal{P}$, $a_l = max_k(A_L)$, $a_r = max_k(A_R)$ and $c_j = min_k(c_i, \dots, c_m)$. Then it either holds that $c_j \succsim_k a_r$ or it holds that $c_j \succsim_k a_l$. This means that, in every vote, the remaining candidates $(c_i, \dots, c_m)$ are all either at least as large as a_r or all at least as large as a_l.*

Proof. Without loss of generality we assume that a_l is placed before a_r. Towards a contradiction assume that $a_r \succ_k c_j$ and $a_l \succ_k c_j$. Let us consider the algorithm at the point when a_r was placed ($c_{i'} = a_r$ and $i' < i$). We will show that (R1) is true and thus a_r could not have been placed on the right-hand side. Recall rule (R1):

$$c_{i'} \succ_k min_k(C_{>i'}) \text{ and } max_k(A_L) \succ_k min_k(C_{>i'})$$

Since $a_r = c_{i'} \succsim_k c_j \succsim_k min_k(C_{>i'})$ and $max_k(A_L) = a_l \succ_k c_j \succsim_k min_k(C_{>i'})$, (R1) is true. □

Lemma 8.8. *We consider the Unguided Algorithm at any given point during its runtime. In particular, we consider the sets A_R and $\{c_i, \dots, c_m\}$. Let $V_k \in \mathcal{P}$ and $c_j = max_k(c_i, \dots, c_m)$. Furthermore, let $a, a' \in A_R$ such that candidate a has been placed on A_R before a'. Then it either holds that $a' \succsim_k c_j$ or it holds that $a' \succsim_k a$.*

Proof. We consider the algorithm at the point where a' was placed on the right-hand side, i.e., in A_R. At this point, condition (R2) has to be false. The fact that $a' \succsim_k c_j$ or $a' \succsim_k a$ holds is a direct consequence of (R2) being false. □

Proposition 8.9. *The Guided Algorithm (Algorithm 5) is correct, i.e., it outputs an axis if and only if the given preference profile is single-peaked and, furthermore, $\mathcal{P}$ is single-peaked with respect to any axis that is returned by the algorithm.*

Proof. We first show that if an axis A is found, the profile $\mathcal{P}$ is single-peaked with respect to A. Towards a contradiction assume that there is a vote $V \in \mathcal{P}$ that is not single-peaked with respect to A. This means that there are three candidates a, b, c with order $a < b < c$ on A, $a \succ b$ and $c \succ b$. We have to distinguish six cases of how a, b, c are ordered by the guiding vote:

- $a \prec b \prec c$ (a is placed first, then b, then c – other candidates in arbitrary order): Let us consider the algorithm at the point when b is being placed, i.e., $b = c_i$, and when the conditions for vote V are being checked. It holds that either $a \in A_L$ or $a \in A_R$. Observe that in the first case Condition (L2) is satisfied since $a \succ b$ and $c \succ b$. Consequently, b has to be placed on the left side (*right* = *false*). Then it holds that $a < c < b$ on the axis generated by the algorithm which contradicts our assumption that $a < b < c$ holds. In the case that $a \in A_R$ Condition (R2) is satisfied. This leads to a contradiction by the same argument.

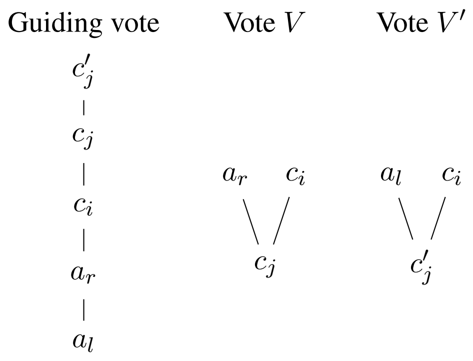


Figure 8.4: Condition (L1) and Condition (R1)

- $c \prec b \prec a$: This case is analogous.
- $a \prec c \prec b$: Now we consider the point where c is being placed, i.e., $c = c_i$, and when the conditions for vote V are being checked. It holds that either $a \in A_L$ or $a \in A_R$. Observe that in the first case Condition (R1) is satisfied and hence c has to be placed on the left side ($right = false$). Then it holds that $a < c < b$ on the axis generated by the algorithm which contradicts our assumption that $a < b < c$ holds. In the case that $a \in A_R$ Condition (L1) is satisfied. This leads to a contradiction by the same argument.
- $c \prec a \prec b$: This case is analogous to the previous one.
- $b \prec c \prec a$ or $b \prec a \prec c$: Since we assume that $a < b < c$ holds on A, these two cases are not possible.

For the other direction, let us show that if the algorithm returns `not_single_peaked`, then the profile $\mathcal{P}$ is not single-peaked. First, let us observe under what conditions the algorithm returns `not_single_peaked`. There are four cases: Either Condition (R1) and (L1), (R1) and (L2), (R2) and (L1) or Condition (R2) and (L2) hold. These pairs of conditions may either hold for the same vote or for two distinct votes; we denote these two votes V and V' although it might be that these two are the same.

- While placing c_i, Condition (L1) holds for some vote V and Condition (R1) holds for some vote V':
  We have the following five candidates in these conditions: $a_r = max_V(A_R)$, c_i, $c_j = min_V(C_{>i})$ in Condition (L1) and $a_l = max_{V'}(A_L)$, c_i, $c'_j = min_{V'}(C_{>i})$ in Condition (R1). In Figure 8.4, the known information about the votes V and V' is shown. Since Condition (R1) and (L1) are symmetrical, we can assume without loss of generality that a_l is placed before a_r and c_j before c'_j. Thus, the guiding vote is as shown in the figure. There are four types of axes possible that are compatible with this guiding vote. (The order of candidates in sets is arbitrary.)

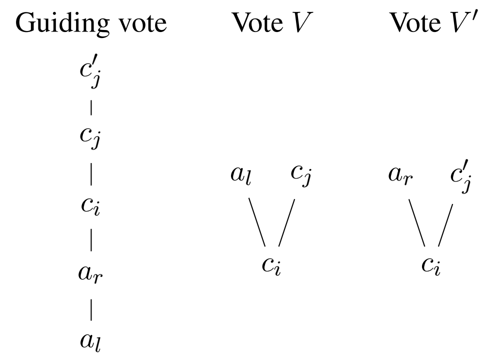


Figure 8.5: Condition (L2) and Condition (R2)

- $\left\langle a_l < c_i < \{c_j, c_j'\} < a_r \right\rangle$: Vote V is not single-peaked with respect to any axis of this type (or their reverse).
- $\left\langle a_l < \{c_j, c_j'\} < c_i < a_r \right\rangle$: Vote V' is not single-peaked with respect to any axis of this type (or their reverse).
- $\left\langle a_l < a_r < \{c_j, c_j'\} < c_i \right\rangle$: Vote V is not single-peaked with respect to any axis of this type (or their reverse).
- $\left\langle a_l < a_r < c_i < \{c_j, c_j'\} \right\rangle$: Consider vote V and Lemma 8.7. Since $a_r \succ c_j$ it has to hold that $c_j \gtrsim a_l$ and $c_i \gtrsim a_l$. Since $c_i \succ c_j$ we know that $c_i \succ a_l$. Also, since $a_r \succ c_j$ we know that $a_r \succ a_l$. Thus, the candidates a_l, a_r, c_i form a v-valley for vote V.

Since these are all the possible axes, we can conclude that the profile is not single-peaked.

- While placing c_i, Condition (L2) holds for some vote V and Condition (R2) holds for some vote V':
  This case is similar to the previous one. In particular, we use the same candidate variables. In Figure 8.5, the known information about the votes V and V' is shown. There are four types of axes possible that are compatible with this guiding vote. (The order of candidates in sets is arbitrary.)

  - $\left\langle a_l < c_i < \{c_j, c_j'\} < a_r \right\rangle$: Vote V is not single-peaked with respect to any axis of this type (or their reverse).
  - $\left\langle a_l < \{c_j, c_j'\} < c_i < a_r \right\rangle$: Vote V' is not single-peaked with respect to any axis of this type (or their reverse).
  - $\left\langle a_l < a_r < \{c_j, c_j'\} < c_i \right\rangle$: Consider vote V and Lemma 8.7. Since $a_l \succ c_i$ it has to hold that $c_j \gtrsim a_r$ and $c_i \gtrsim a_r$. Since $c_j \succ c_i$ we know that $c_j \succ a_r$. Also, since

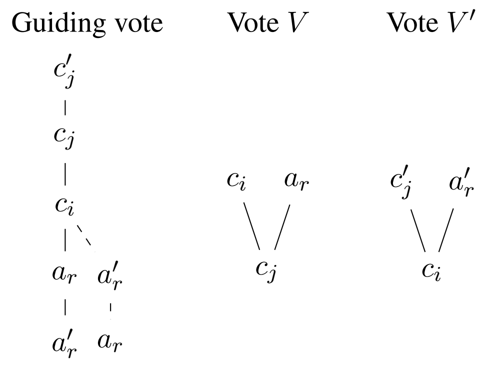


Figure 8.6: Condition (L1) and Condition (R2)

$a_l \succ c_i$ we know that $a_l \succ a_r$. Thus, the candidates a_l, a_r, c_j form a v-valley for vote V.

– $\left\langle a_l < a_r < c_i < \{c_j, c_j'\} \right\rangle$: Vote V' is not single-peaked with respect to any axis of this type (or their reverse).

- While placing c_i, Condition (L1) holds for vote V and Condition (R2) holds for vote V': We have the following five candidates in these conditions: $a_r = \mathit{max}_V(A_R)$ c_i, $c_j = \mathit{min}_V(C_{>i})$ in Condition (L1) and $a_r' = \mathit{max}_{V'}(A_R)$, c_i, $c_j' = \mathit{min}_{V'}(C_{>i})$ in Condition (R2). In Figure 8.6, the known information about the votes V and V' is shown. In the following arguments it is irrelevant which of c_j and c_j' is placed first. However, for a_r and a_r' this is relevant. We will consider both cases. There are four types of axes possible that are compatible with this guiding vote. (The order of candidates in sets is arbitrary.)

  – $\left\langle \{a_r, a_r'\} < \{c_j, c_j'\} < c_i \right\rangle$: Vote V is not single-peaked with respect to any axis of this type (or their reverse).

  – $\left\langle \{a_r, a_r'\} < c_i < \{c_j, c_j'\} \right\rangle$: Vote V' is not single-peaked with respect to any axis of this type (or their reverse).

  – $\left\langle a_r < \{c_j, c_j'\} < c_i < a_r' \right\rangle$: Both V and V' are not single-peaked with respect to any axis of this type (or their reverse).

  – $\left\langle a_r < c_i < \{c_j, c_j'\} < a_r' \right\rangle$: Here we have to distinguish whether a_r or a_r' is placed first.

    * Let us assume that a_r is placed before a_r'. We apply Lemma 8.8 to vote V. This yields that either $a_r' \succsim c_i$ or that $a_r' \succsim a_r$. If $a_r' \succsim c_i$ holds then $a_r' \succsim c_i \succ c_j$ holds. If $a_r' \succsim a_r$ holds then $a_r' \succsim a_r \succ c_j$ holds. In both cases vote V forms a valley on the candidates a_r', a_r, c_j.
    * Let us assume that a_r' is placed before a_r. We apply Lemma 8.8, this time to vote V'. This yields that either $a_r \succsim' c_j'$ or that $a_r \succsim' a_r'$. If $a_r \succsim' c_j'$ holds then

$a_r \precsim' c_j' \succ' c_i$ holds. If $a_r \precsim' a_r'$ holds then $a_r \precsim' a_r' \succ' c_i$ holds. In both cases vote V' forms a valley on the candidates a_r', a_r, c_i.

We see that in both cases either V or V' is not single-peaked with respect to any axis of this type (or their reverse).

- While placing c_i, Condition (L2) holds for some vote and Condition (R1) holds for some vote:
  This can be shown analogously to the previous case since (L1) and (L2) are symmetrical as well as (R1) and (R2), cf. Figure 8.3.

We have shown that if the algorithm returns `not_single_peaked` then the profile $\mathcal{P}$ is indeed not single-peaked. □

We conclude this section with a lemma showing that we can weaken the total order requirement: it suffices that the guiding vote is given implicitly in the profile.

Lemma 8.10. *Let $C = \{c_1, c_2, \ldots, c_m\}$ and $T = \langle c_1 < c_2 < \ldots < c_m \rangle$ be a total order on C with the following property: for each $i \in \{1, \ldots, m\}$, it holds that there is a vote $V \in \mathcal{P}$ such that c_i is the unique last ranked candidate in $V[\{c_i, c_{i+1}, \ldots, c_m\}]$. If $\mathcal{P}$ is a single-peaked profile, then $\mathcal{P}$ is also single-peaked if the total order T is added to it as a vote.*

It is computationally easy to find such an implicitely given guiding vote: Look for a vote with a unique last ranked candidate. This candidate is ranked last in the guiding vote. Remove this candidate from the profile and repeat this step to obtain the second-to-last element in the guiding vote, etc.

8.5 A 2-SAT Based Algorithm

Theorem 8.5 and Theorem 8.6 leave open the case of profiles of local weak orders which contain at least one total order. Here, we show that this case is polynomial time solvable as well.

Theorem 8.11. *If the profile contains a total order, the* Local Weak Order Single-peaked Consistency *problem can be solved in $\mathcal{O}(n \cdot m^3)$ time.*

We encode a Local Weak Order Single-peaked Consistency instance in a 2-Sat instance. The 2-Sat problem asks whether a Boolean formula of the form $(a \vee b) \wedge (\neg a \vee c) \wedge \ldots$ (each clause has size two) is satisfiable. Solving 2-Sat requires only linear time [11]. The boolean variables in our instance correspond to pairs of candidates, i.e., for each $a, b \in C$ we have a variable ab. The intended meaning of these variables is that $ab = \mathit{true}$ if and only if a is left of b on the axis. Now, for each vote V and triple $a, b, c \in C$, if $a \succ b$ and $c \succ b$ (a, b, c form a v-valley), we add the clauses $(ba \vee cb)$ and $(ab \vee bc)$ to the 2-Sat instance. These clauses correspond the requirement that b must not be placed between a and c. Finally, we add for each pair of variables a, b the clauses $(ab \vee ba)$ and $(\neg ab \vee \neg ba)$ (corresponding to the exclusive or operator). Solving the 2-Sat instance either yields the information that the instance is not satisfiable or a true/false assignment to the variables. In the first case, the profile is not single-peaked. In the second case, we obtain a relation $A = \{(a, b) : ab = \textit{true}\} \cup \{(a, a) : a \in C\}$ which is our wanted axis (as shown in Lemma 8.12). Since the instance contains at most $\mathcal{O}(n \cdot m^3)$ clauses, we obtain the stated runtime.

Lemma 8.12. *The axis A, as returned by the* 2-SAT *algorithm, is a total order and $\mathcal{P}$ is single-peaked with respect to A.*

Proof. It is straight-forward to verify that A is reflexive, antisymmetric and total. Towards a contradiction assume that A is not transitive, i.e., there exist three candidates a, b, c such that $\{(a, b), (b, c), (c, a)\} \subseteq A$. Thus, $ab = bc = ca = \textit{true}$. Let V be a total contained in $\mathcal{P}$ (there exists at least one). We distinguish three cases:

- The last ranked candidate of a, b, c in V is b: By our construction, it has to hold that $(ba \vee cb)$ – which is not the case.
- The last ranked candidate of a, b, c in V is a: It has to hold that $(ba \vee ac)$ – which is not the case.
- The last ranked candidate of a, b, c in V is c: It has to hold that $(cb \vee ac)$ – which is not the case.

Thus, A is transitive. It remains to show that $\mathcal{P}$ is single-peaked with respect to A. Assume that there is a valley $a \succ b$, $c \succ b$ in some vote and it holds that $\{(a, b), (b, c)\} \subseteq A$. Due to this valley, our 2-SAT instance contains the clause $(ba \vee cb)$. Thus, $(b, a) \in A$ or $(c, b) \in A$ and thus $ba = \textit{true}$ or $cb = true$. This contradicts our requirement that for every pair of variables x, y not both xy and yx can be true. □

Both the 2-SAT based algorithm and the Guided Algorithm rely on the guiding vote. In the next section, we will consider profiles that do not have a guiding vote.

8.6 The Unguided Algorithm

Here, we present a polynomial-time algorithm (Algorithm 6) that, in contrast to the Guided Algorithm, is not dependent on a guiding vote. We therefore refer to it as the Unguided Algorithm. The Unguided Algorithm is applicable to top orders. We assume the input preference profile to be *connected*: Let us consider the ranked candidates in a top order to be a hyperedge of a hypergraph with candidates as vertices. A profile of top orders is called connected if this graph has only one connected component. This assumption does not limit the applicability: if two or more connected components exist in this graph, we can use the algorithm for each component (i.e., its respective candidates and votes) and concatenate the resulting axes in arbitrary order.

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

Algorithm 6: The Unguided Algorithm

Input: A set of candidates C and a connected preference profile of top orders $\mathcal{P} = (V_1, \dots V_n)$.
Output: An axis A or *not_single_peaked*.

1 **foreach** $c_{start} \in C$ **do**
2     $A \leftarrow \langle c_{\text{start}} \rangle$
3   **for** $i \leftarrow 1 \dots m$ **do**
4   **foreach** $V \in$ `VotesWithPeak`(a_i) **do**
5   **if** $A \oplus V =$ `incompatible` **then**
6   Continue with next $c_{\text{start}} \in C$ in Line 1.
7   **else** $A \leftarrow A \oplus V$
8
9   **if** $|A| = i$ *and* $i < m$ **then**
10             $V \leftarrow$`IntersectingVote`(A)
11   **if** $a_i \notin V$ **then**
12   Continue with next $c_{\text{start}} \in C$ in Line 1.
13   Let x be a new candidate not in C.
14             $C' \leftarrow \{c \in V \mid c \succ a_i\} \cup \{a_i, x\}$
15             $S \leftarrow \emptyset$
16   **for** $k \leftarrow 1 \dots n$ **do**
17                 $V'_k \leftarrow$ `RepTop`$(V_k, C \setminus (A \cup C'), x)$
18                 $S \leftarrow S \cup \{V'_k[C']\}$
19             $A' \leftarrow$`GuidedSP`$(S, V[C'], a_i, x)$
20   **if** $A' =$`not_single_peaked` **then**
21   Continue with next $c_{\text{start}} \in C$ in Line 1.
22   **else** $A \leftarrow A < A'[C' \setminus \{x\}]$
23
24   **return** A
25 **return** `not_single_peaked`

employ the loop in Line 3. In this loop (variable i) we infer from the already placed candidate a_i (the i-th candidate on A from left) the candidate a_{i+1} (or even more candidates further to the right).

The Lines 4 to 7 are based on the following observation: Let us assume that at a certain point $A = \langle c_1 < c_2 < c_3 \rangle$ and $V = \langle c_3 \succ c_2 \succ c_4 \succ c_5 \succ \bullet \rangle \in \mathcal{P}$. Since c_3, the peak of V, is already contained in A, there is only one compatible extension of A: $\langle c_1 < c_2 < c_3 < c_4 < c_5 \rangle$. We formalize this extension operation with the $\oplus$ operator:

Definition 8.5. *Let A be an incomplete axis and V a top order. Furthermore, let $V[C \setminus A] = \left\langle c'_1 \succ c'_2 \succ \dots \succ c'_j \succ \bullet \right\rangle$. We define $A \oplus V = \left\langle A < c'_1 < c'_2 < \dots < c'_j \right\rangle$ if V is single-peaked with respect to this axis and $A \oplus V =$* `incompatible` *otherwise.*

The correctness (and necessity) of the $\oplus$ operator is a consequence of the following lemma.

Lemma 8.13. *Let A be an incomplete axis and V a vote that satisfies the conditions in Definition 8.5. If B is an extension of A and V is single-peaked with respect to B, then B is also an extension of $A \oplus V$.*

Proof idea. If B were not an extension of $A \oplus V$, then V would contain a valley with respect to B. $\square$

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

It might be the case that the candidate a_{i+1} has not yet been determined after these steps. The Lines 9 to 22 deal with this case. Since the election is connected there has to be at least one vote that ranks both a candidate on A and a candidate that has not been placed yet. The procedure $\texttt{IntersectingVote}$ in Line 10 returns such a vote V with $A \cap V \neq \emptyset$ and $V \setminus A \neq \emptyset$. For such a vote V it holds that $\text{peak}(V) \notin A$. If $\text{peak}(V)$ were contained in A, then V would have been already considered in the first part of the algorithm (Lines 4 to 7). If V does not contain a_i (and thus a_i is ranked last in V), A cannot be extended to a single-peaked axis. This procedure can be efficiently precomputed in such a way that it can requires only $\mathcal{O}(m)$ time to provide an answer. Details can be found in the proof of Theorem 8.14.

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

Example. Let $A = \langle c_1 \rangle$, $V_1 = \langle c_2 \succ c_3 \succ c_1 \succ \bullet \rangle$ and $V_2 = \langle c_3 \succ c_4 \succ \bullet \rangle$. The vote V_1 intersects A and hence $C' = \{c_1, c_2, c_3\}$. We employ the Guided Algorithm and might obtain $A' = \langle c_1 < c_3 < c_2 \rangle$.[2] Now observe that $A \oplus A' = A'$ can no longer be extended in a way that it is single-peaked for V_2. This would have been possible if c_3 had been chosen as the rightmost candidate in A'.

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

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

candidate, we ensure that A' is chosen under consideration of all votes with ranked candidates not in C'.

Example (continued)**.** We apply `RepTop`(V, D, x) to the votes V_1 and V_2 with candidate sets $C' = \{c_1, c_2, c_3, x\}$ and $D = \{c_4\}$. We obtain the votes $V_1'[C'] = \langle c_2 \succ c_3 \succ c_1 \succ x\rangle$ and $V_2'[C'] = \langle c_3 \succ x \succ \bullet\rangle$. Now, we can only obtain the axis $\langle c_1 < c_2 < c_3 < x\rangle$.

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

Theorem 8.14. *The* TOP ORDER SINGLE-PEAKED CONSISTENCY *problem can be solved in* $\mathcal{O}(m^2 \cdot n)$ *time.*

Proof. The runtime is achieved by precomputing the functions `VotesWithPeak` as well as `IntersectingVote`. The function `VotesWithPeak` is stored as a list of lists containing each vote exactly once. It can be computed in $\mathcal{O}(m \cdot n)$ time.

The function `IntersectingVote`(A) returns a vote V with $A \cap V \neq \emptyset$ and peak$(V) \notin C$. We show that it suffices to compute a list of $2m$ votes to answer `IntersectingVote` function calls in constant time. Let us first make the following observation: Let $c \in C$. Consider the set of votes for which the sets $\{c' \in C \mid c' \succ c\}$ are maximal (with respect to $\subseteq$). If we consider a single-peaked axis, then candidates in such a set have to form a contiguous subsequence either directly left or directly right of c. Since these sets are maximal, only two of them can exist (assuming single-peakedness). Consequently, we compute these maximal sets for each candidate. If three or more exist for one candidate, we can terminate the algorithm already at this point. Also, if two maximal sets have a non-empty intersection, the algorithm terminates. (The candidates in the intersection would have to lie left and right of c). Otherwise we store the (at most) two corresponding votes for each candidate.

Let $A = \langle c_1 < \ldots < c_{i-1} < c_i\rangle$, i.e., c_i is the rightmost candidate in the incomplete axis A. The function call `IntersectingVote`(A) can now be answered by considering the one or two maximal votes for c_i. The function simply returns the vote where c_{i-1} is not ranked higher than c_i. (It might be that both votes do not rank c_{i-1} higher than c_i. In this case A cannot be extended to a single-peaked axis, but this is going to be detected by algorithm. Any of the two axes can be returned.)

It remains to observe that finding the (at most two) maximal votes for a candidate c requires $\mathcal{O}(m \cdot n)$ time. This has to be done for every candidate and consequently this preprocessing requires $\mathcal{O}(m^2 \cdot n)$ time.

We can now analyze the runtime of the algorithm. The main loop (Line 1) iterates over all m candidates. The loop in Line 4 iterates over every vote at most once. Consequently, the $\oplus$ operator is applied at most n times. Since $A \oplus V$ can be computed in $\mathcal{O}(m)$ time, the Lines 4 to 7 have a total runtime of $\mathcal{O}(m^2 \cdot n)$.

It remains to determine the runtime of the Lines 9 to 22. Due to the preprocessing of the `IntersectingVote` procedure we can obtain V in constant time. The set S can be generated in $\mathcal{O}(|C'| \cdot n)$ time. Applying the Guided Algorithm requires $\mathcal{O}(|C'| \cdot n)$ time as well (Theorem 8.6). Observe that after applying the Guided Algorithm the candidates in C' are placed on the axis. Consequently, the Guided Algorithm is always applied to a disjoint set of candidates (except maybe a_i). Hence for a fixed $c_{\text{start}} \in C$, the total runtime of the Guided Algorithm is $\mathcal{O}(m \cdot n)$. Taking the loop in Line 1 into account, we obtain a total runtime of $\mathcal{O}(m^2 \cdot n)$.

Let us now show that the Unguided Algorithm (Algorithm 6) is correct, i.e., it outputs an axis if and only if the given preference profile is single-peaked and, furthermore, $\mathcal{P}$ is single-peaked with respect to any axis that is returned by the algorithm. If the algorithm outputs an axis, it is certainly single-peaked since this is tested for every vote in Line 5. By the same argument, one can conclude that if the profile is not single-peaked, the algorithm returns *not_single_peaked*.

It remains to prove that the algorithm always returns a valid axis in case of a single-peaked profile. Let us consider the algorithm at the time when c_{start} is the leftmost candidate of a valid axis. We show that the algorithm will find a complete axis with c_{start} as leftmost candidate. First, observe that for every i (Line 3) either `VotesWithPeak`(a_i) is not empty or the condition in Line 9 is true. If this were not the case, the profile would not be connected. In the first case, the $\oplus$ operator adds candidates to the axis in the only possible way (Lemma 8.13). Hence, the axis necessarily has to be extended in that way. In the second case, the Guided Algorithm is applied and the axis is extended by the candidates in C'. It remains to verify that the resulting axis A is single-peaked for all votes with a non-empty intersection with C'. This is guaranteed by the x element, which ensures that candidates outside of $A \cup C'$ are taken into account. $\square$

8.7 Scoring Protocols

We would like to mention one particular application of the Unguided Algorithm concerning single-peaked scoring protocols. Scoring protocols are specified by a scoring vector given as $(\alpha_1, \dots, \alpha_m)$. A vote $c_1 \succ \dots \succ c_m$ gives α_1 points to c_1, α_2 points to c_2, etc. The winner candidate is determined by summing over all votes. Often scoring vectors of the type $(\alpha_1, \dots, \alpha_k, 0, \dots, 0)$ with $\alpha_1 > \dots > \alpha_k > 0$ are considered. For such scoring vectors, top orders (with k ranked candidates) constitute *full* information. It is therefore debatable whether the input may be considered to be given as a profile of total orders, as total orders contain problem-irrelevant information. This is relevant for single-peaked profiles. For example, Brandt et al. [41] study the constructive coalition weighted manipulation problem for scoring protocols in single-peaked elections. The authors consider the axis to be part of the input (for good reasons as explained in their paper). The computation of such an axis with existing algorithms is possible only if preferences are specified by total orders and thus contain problem-irrelevant information. If only relevant information is given, i.e., the input consists of top orders, an algorithm such as the Unguided Algorithm is required.

X	general	guiding vote
PARTIAL	NP-c (Cor 8.4)	NP-c (Thm 8.5)
LOCAL WEAK	NP-c (Thm 8.3)	poly (Thm 8.11)
WEAK	open	poly (Thm 8.6)
TOP	poly (Thm 8.14)	poly (Thm 8.6)
TOTAL	poly[3]	poly[3]

Table 8.1: Overview of the complexity results for $\mathcal{T}$ ORDER SINGLE-PEAKED CONSISTENCY

8.8 Summary

In this chapter we have analyzed the $\mathcal{T}$ SINGLE-PEAKED CONSISTENCY problem for $\mathcal{T} \in$ {PARTIAL ORDER, LOCAL WEAK ORDER, WEAK ORDER, TOP ORDER, TOTAL ORDER}. An overview of the results are displayed in Table 8.1. Despite the NP-completeness of PARTIAL ORDER SINGLE-PEAKED CONSISTENCY, we have found four fast algorithms for plausible application scenarios. The Guided Algorithm and the 2-SAT based algorithm require a guiding vote. Such an order is likely to exist for large preference profiles. In the case that top orders are elicited, a guiding vote might not exist. Here the Unguided Algorithm is applicable. In addition, we have found that PARTIAL ORDER SINGLE-PEAKED CONSISTENCY is solvable in polynomial time if the axis is already part of the input. We therefore believe to have succeeded in covering a large spectrum of possible application scenarios with our algorithms.

[3]This is a result by Bartholdi and Trick [23].

CHAPTER 9

Connections between Structure in Permutation Patterns and in Preferences

This chapter is based on joint work with Marie-Louise Bruner that is partially published in *The Likelihood of Structure in Preference Profiles* [48].

Here, we establish a strong link between the concept of configuration containment in profiles and the pattern containment in permutations. This link enables us to approach the following two questions concerning preferences with the help of methods and results concerning permutation patterns.

- How likely is it that a preference profile belongs to a restricted domain?
- What is the computational complexity of testing whether a preference profile belongs to a restricted domain?

The first question, despite the extensive literature on domain restrictions, has not received much attention so far. There are two experimental studies on that topic: Mattei, Forshee and Goldsmith [116] report that in their data sets almost no evidence for the single-peaked restriction was found. Similarly, Sui, Francois-Nienaber and Boutilier [135] report also no occurences of the single-peaked restriction in their data sets. However, they found that the preferences in their data set are close to being 2D single-peaked.

Our contribution, in contrast, is of theoretical nature. We employ combinatorial methods to study the likelihood of structure in preference profiles chosen according to the Independent Culture assumption, i.e., all votes are equally likely to appear in the profile. Our result applies to domain restrictions that can be characterized by a set of configurations (cf. Section 7.1) where one of those configurations has cardinality two, i.e., one of those configurations consists of two conditions. Thus, our results are applicable to the single-peaked, single-caved, group-separable and 1D Euclidean [55, 108] restriction. We show that these domain restrictions are very unlikely to appear in a random profile chosen according to the Impartial Culture assumption. More precisely, while the total number of profiles with n votes and m candidates is equal to $(m!)^n$, the number of profiles belonging to such a domain restriction can be bounded by $m! \cdot c^{nm}$ for some constant c. This theorem is obtained by utilizing the Marcus–Tardos theorem [115], a famous result about permutation patterns.

We also approach the second question, concerning the computational complexity of detecting domain restrictions, by studying configurations. If a domain restriction can be characterized by forbidden configurations (as it is the case, for example, for all domain restrictions studied in Chapter 7), detecting these domain restrictions is computationally equivalent to the following problem:

CONFIGURATION CONTAINMENT	
Instance:	A profile $\mathcal{P}$ and a configuration Φ with conditions in disjunctive normal form.
Question:	Is Φ contained in $\mathcal{P}$?

Here, we require that the configuration Φ consists of conditions in disjunctive normal form. This is the case for all domain restrictions studied in this paper. In addition, this circumvents the problem that it is NP-complete even to decide whether a Boolean condition is satisfiable. A hardness result based on this observation would not be satisfactory and, hence, we make the reasonable assumption that all conditions are in disjunctive normal form, for which testing satisfiability is tractable.

Despite this restriction, we show that CONFIGURATION CONTAINMENT is NP-hard if $|\mathcal{P}| \geq 2$ and $|\Phi| \geq 2$ and (trivially) polynomial-time solvable otherwise. We also study the parameterized complexity of CONFIGURATION CONTAINMENT, where we prove a parameterized complexity dichotomy. These results make use of complexity results from permutation patterns, in particular Theorem 4.7. Our results indicate that the algorithm for detecting configurations presented in Chapter 7 (Proposition 7.1) cannot be substantially improved, i.e., a universally applicable fpt algorithm is not possible.

9.1 Applying the Marcus–Tardos Theorem to Domain Restrictions

In this section, we make use of the Marcus–Tardos Theorem to obtain combinatorial results about preferences. To be able to speak about the number of profiles avoiding a given configuration, we have to fix the names of candidates. Thus, we assume that if $|C| = m$ then $C = \{1, \dots, m\}$. An (n, m)-profile is then a profile with n votes and with candidate set $C = \{1, \dots, m\}$.

Let us start with two definitions. First, building upon the definitions in Section 7.1, we define completions of configurations.

Definition 9.1. *Given a configuration $\Phi = (\phi_1, \dots, \phi_s)$, we say that a configuration $\Phi' = (\phi'_1, \dots, \phi'_s)$ is a* completion *of Φ if for every $i \in [s]$ it holds that ϕ'_i has exactly one model,* *this model is a total order and it is also a model of ϕ_i. If a condition ϕ has a unique model, let mod(ϕ) be this model.*

The second definition establishes the connection between total orders and permutations.

Definition 9.2. *Recall that $T(i)$ denotes the i-th largest element with respect to T. Every pair T_1, T_2 of total orders on a set with m elements can be identified with the m-permutation $p(T_1, T_2) := \{i \mapsto j : T_1(j) = T_2(i)\}$.*

For example, if $T_1 = c > a > b$ and $T_2 = b > a > c$, we have $p(T_1, T_2) = 321$. Note that $p(T_1, T_2) = p(T_2, T_1)^{-1}$ holds in general.

The following lemma establishes a link between configuration containment in profiles and pattern containment in permutations.

Lemma 9.1. *Let $\Phi = (\phi_1, \phi_2)$ be a completion of some arbitrary configuration and let $C_1 =$ mod(ϕ_1) and $C_2 =$ mod(ϕ_2). The configuration Φ is contained in the profile $\mathcal{P} = (V_1, V_2)$ if and only if the permutation $\pi = p(C_1, C_2)$ or the permutation $\pi^{-1} = p(C_2, C_1)$ is contained in $p(V_1, V_2)$.*

Proof. In order to alleviate notation, we will assume in the following that the candidate set $C = \{1, 2, \ldots, m\}$ and $X(\Phi) = \{1, 2, \ldots, k\}$ (the variables used in the conditions of Φ).

"$\leftarrow$" We can assume without loss of generality that C_1 is $1 > 2 > \cdots > k$ and V_1 is $1 \succ 2 \succ \cdots \succ m$. If π is contained in $p(V_1, V_2)$ as witnessed by a matching M, then $V_1 \models_M \phi_1$ and $V_2 \models_M \phi_2$ (cf. Definition 7.1). If π^{-1} is contained in $p(V_1, V_2)$ as witnessed by a matching M, then $V_1 \models_M \phi_2$ and $V_2 \models_M \phi_1$.

"$\rightarrow$" Let Φ be contained in $\mathcal{P}$ with $V_{i_1} \models_\xi \phi_1$ and $V_{i_2} \models_\xi \phi_1$. Note that either $i_1 = 1$ and $i_2 = 2$ or that $i_1 = 2$ and $i_2 = 1$. Without loss of generality we assume that ϕ_1 is $1 > 2 > \cdots > k$. Note that renaming the candidates (in C) does not change whether Φ is contained in $\mathcal{P}$. Thus, it is safe to rename the candidates according to i_1 and i_2: If $i_1 = 1$ and $i_2 = 2$, let V_1 be $1 \succ 2 \succ \cdots \succ n$. Since $V_1 \models_\xi \phi_1$, ξ is monotonic. It is easy to verify that ξ is a matching from π into $p(V_1, V_2)$. On the other hand, if $i_1 = 2$ and $i_2 = 1$, let V_2 be $1 \succ 2 \succ \cdots \succ n$. Now, ξ is a matching from π into $p(V_2, V_1) = (p(V_1, V_2))^{-1}$. This is equivalent to ξ being a matching from π^{-1} into $p(V_1, V_2)$. □

As of now, we shall denote by $S_m(\pi_1, \ldots, \pi_l)$ the cardinality of the set of m-permutations that avoid the permutations $\pi_1, \ldots, \pi_l$.

Corollary 9.2. *Let $\Phi = (\phi_1, \phi_2)$ be a completion of some arbitrary configuration. Furthermore, let V_1 be a vote on $m \geq 1$ candidates. Then the number of votes V_2 such that the profile $\mathcal{P} = (V_1, V_2)$ avoids the configuration Φ is equal to $S_m(\pi, \pi^{-1})$, where $\pi = p($mod$(\phi_1),$ mod$(\phi_2))$.*

Proof. Lemma 9.1 tells us that $\mathcal{P} = (V_1, V_2)$ avoids the configuration Φ if and only if the permutation $p(V_1, V_2)$ avoids both the patterns π and π^{-1}. Moreover, for the fixed total order V_1 and a fixed m-permutation σ, there is a single total order V_2 such that $p(V_1, V_2) = \sigma$. Thus the number of votes V_2 such that $p(V_1, V_2)$ avoids π and π^{-1} and equivalently the number of votes V_2 such that $\mathcal{P}$ avoids Φ is equal to $S_m(\pi, \pi^{-1})$, the number of m-permutations avoiding π and π^{-1}. □

From this follows a very general result that is applicable to any set of configurations that contains at least one configuration of cardinality two.

Theorem 9.3. *Let $a(n, m, \Gamma)$ be the number of (n, m)-profiles avoiding a set of configurations Γ. Let $k \geq 2$. If a set of configurations Γ contains a $(2, k)$-configuration $\Phi = (\phi_1, \phi_2)$, then it holds for all $n, m \geq 1$ that $a(n, m, \Gamma) \leq m! \cdot c_k^{(n-1)m}$, where c_k is a constant depending only on k.*

Proof. Instead of Φ, we will consider a completion of Φ; let us call this completion Φ'. This can be done without loss of generality, since $a(n, m, \{\Phi\}) \leq a(n, m, \{\Phi'\})$.

We want to determine the number of (n, m)-profiles avoiding Φ'. Let us start by choosing the first vote V_1 of the profile at random. For this there are $m!$ possibilities. When choosing the remaining $(n-1)$ votes $V_2, \ldots, V_n$, we have to make sure that no selection of two votes contains the forbidden configuration Φ'. If we relax this condition and only demand that none of the pairs (V_1, V_i) for $i \neq 1$ contains the forbidden configuration, we clearly obtain an upper bound for $a(n, m, \{\Phi'\})$.

Now Corollary 9.2 tells us that there are – under this relaxed condition – $S_m(\pi, \pi^{-1})$ choices for every V_i where $\pi := p(C_1, C_2)$. Thus we have the following upper bound:

$$a(n, m, \{\Phi'\}) \leq m! S_m(\pi, \pi^{-1})^{n-1} \leq m! S_m(\pi)^{n-1}, \tag{9.1.1}$$

where the second inequality follows since all permutations avoiding both π and π^{-1} clearly avoid π.

Now we apply the famous Marcus–Tardos theorem [115]: For every permutation π of length k there exists a constant c_k such that for all positive integers m we have $S_m(\pi) \leq {c_k}^m$. Putting this together with Equation (9.1.1) and noting that $a(n, m, \{\Phi'\})$ is an upper bound for $a(n, m, \Gamma)$ we obtain the desired upper bound. $\square$

This result shows that forbidding any $(2, k)$-configuration is a very strong restriction on preference profiles. Indeed, $m! \cdot c_k^{(n-1)m}$ is very small compared to the total number of (n, m)-profiles which is $(m!)^n$.

The proof of the Marcus–Tardos theorem provides an explicit exponential formula for the constants c_k, but these constants are far from being optimal. There is an ongoing effort to find exact formulas for $S_m(\pi)$ with fixed π [105].

Let us discuss the implications of this theorem. It is applicable to all configuration definable domain restrictions that contain a configuration of cardinality two. This includes the single-peaked restriction as well as the 1D Euclidean [55, 108] and group separable [16] restriction.

9.2 Computational Results

In this section we study the computational problem of checking whether a configuration is contained in a profile. The results in this section heavily build upon the relation between configuration containment and permutation patterns that was established in the previous section (Lemma 9.1). The algorithm for detecting configurations presented in Chapter 7 (Proposition 7.1) has a runtime of $\mathcal{O}(||\Phi||nm^t)$, where $t = |X(\Phi)|$. The main goal of this section is to determine whether this runtime can be substantially improved. The strongest improvement would be to find a polynomial-time algorithm. A weaker but still significant improvement would be an fpt algorithm with respect to the parameter t, i.e., to find an algorithm with a runtime of, say, $\mathcal{O}(||\Phi|| \cdot 2^t \cdot nm)$. As a first result, we prove that a polynomial time algorithm does not exist unless $\mathsf{P} = \mathsf{NP}$.

Theorem 9.4. *The* CONFIGURATION CONTAINMENT *problem is* NP*-complete, even if* $|\mathcal{P}| = 2$ *and* $|\Phi| = 2$.

Proof. We reduce from the NP-complete PERMUTATION PATTERN MATCHING problem (cf. Chapter 4 and 5). Let σ denote the text permutation and π denote the pattern permutation. We construct a profile consisting of two total orders $\mathcal{P} = (V_1, V_2)$ with $C = [3n+2]$ and a configuration $\Phi = (\phi_1, \phi_2)$ with $X(\Phi) = [2n+k+2]$. The total order V_1 is $1 \succ 2 \succ \ldots \succ 3n+2$; the total order V_2 is defined as

$$\begin{aligned} &2 \succ 4 \succ \cdots \succ 2n+2 \succ 2n+1 \succ 2n-1 \succ \cdots \succ 3 \succ 1 \succ \\ &\succ \sigma(1)+2n+2 \succ \sigma(2)+2n+2 \succ \cdots \succ \sigma(n)+2n+2. \end{aligned}$$

The condition ϕ_1 is $1 > 2 > \ldots > 2n+k+2$; the condition ϕ_2 is defined as

$$\begin{aligned} &2 > 4 > \cdots > 2n+2 > 2n+1 > 2n-1 > \cdots > 3 > 1 > \\ &> \pi(1)+2n+2 > \pi(2)+2n+2 > \cdots > \pi(k)+2n+2. \end{aligned}$$

We show that the configuration (ϕ_1, ϕ_2) is contained in (V_1, V_2) if and only if π is contained in σ.

"$\leftarrow$" Assume that there is a matching M from π into σ. We claim that there exists a function $\xi : [2n+k+2] \to [3n+2]$ such that $V_1 \models_\xi \phi_1$ and $V_2 \models_\xi \phi_2$ (cf. Definition 7.1). Let ξ be defined as

$$\xi(i) = \begin{cases} i & \text{if } i \in [2n+2] \\ 2n+2+M(i-2n-2) & \text{if } i \in [2n+3, 2n+2+k]. \end{cases}$$

Observe that since M is strictly monotone, also ξ is strictly monotone. Also, $V_1 \models_\xi \phi_1$ is an immediate consequence of V_1 and ϕ_1 being monotone. That $V_2 \models_\xi \phi_2$ is more tedious to check but fundamentally a consequence of M being a matching.

"$\rightarrow$" Assume that (ϕ_1, ϕ_2) is contained in (V_1, V_2). Lemma 9.1 implies that either the permutation $p(\mathit{mod}(\phi_1), \mathit{mod}(\phi_2))$ or the permutation $p(\mathit{mod}(\phi_2), \mathit{mod}(\phi_1))$ is contained in $p(V_1, V_2)$. First, let us observe that $p(V_1, V_2)$ is the permutation

$$(2, 4, \ldots, 2n+2, 2n+1, 2n-1, \ldots, 3, 1, \sigma(1)+2n+2, \ldots, \sigma(n)+2n+2)$$

and that this permutation contains an ascending subsequence of length $n+1$ followed by a descending subsequences of length $n+1$. The permutation $p(\mathit{mod}(\phi_2), \mathit{mod}(\phi_1))$ is

$$(2n+2, 1, 2n, 2, 2n-1, 3, \ldots, n+2, n+1, [2n+3, 2n+2+k]),$$

where $[2n+3, 2n+2+k]$ means these elements are in some (not explicitly specified) order. This permutation consists of two interleaving ascending and descending subsequences of length $n+1$ followed by the elements $[2n+3, 2n+2+k]$. If $p(\mathit{mod}(\phi_2), \mathit{mod}(\phi_1))$ was contained in $p(V_1, V_2)$, then $p(V_1, V_2)$ would have to contain two interleaving ascending and descending subsequences of length $n+1$ – just as $p(\mathit{mod}(\phi_2), \mathit{mod}(\phi_1))$ does. Since this is not the case, $p(\mathit{mod}(\phi_1), \mathit{mod}(\phi_2))$ has to be contained in $p(V_1, V_2)$. It follows that π is contained in σ; the matching can be obtained from the corresponding ξ. □

Observe that the restrictions in Theorem 9.4 are as strong as possible in the following sense: Configurations with only one (satisfiable) condition are contained in any profile and thus, in such a case, we have a trivial yes-instance. If $|\mathcal{P}| = 1$, either $|\Phi| = 1$ (and hence we have a yes-instance) or $|\Phi| > 1$, in which case we have a trivial no-instance.

The question we have not answered with this theorem is whether an fpt algorithm for CONFIGURATION CONTAINMENT is possible. Our next theorem shows that such an fpt result is indeed possible for configurations of size two.

Theorem 9.5. *If* $|\Phi| = 2$, *then* CONFIGURATION CONTAINMENT *is in* FPT *with respect to* $t(\Phi) = |X(\Phi)|$.

Proof. The algorithm builds upon the result by Guillemot and Marx that PERMUTATION PATTERN MATCHING can be solved in FPT time parameterized by the length of the pattern permutation [92]. Their algorithm has a runtime of $2^{\mathcal{O}(k^2 \cdot \log k)} \cdot m$, where k is the length of π and m the length of σ. Our algorithm works as follows: For every pair of total orders T_1, T_2 on $X(\Phi)$ with T_1 being a model of ϕ_1 and T_2 being a model of ϕ_2 and every pair of votes $V_1, V_2 \in \mathcal{P}$, we check whether the permutation $p(T_1, T_2)$ or $p(T_2, T_1)$ is contained in $p(V_1, V_2)$. If $p(T_1, T_2)$ or $p(T_2, T_1)$ is contained, we know that (C_1, C_2) is contained in $\mathcal{P}$ (cf. Lemma 9.1). If in none of these combinations a pattern containment is detected, $\mathcal{P}$ avoids Φ. Observe that we have to employ the permutation pattern matching algorithm at most $(t!)^2$ times and thus our algorithm runs in fpt time. □

We now show that Theorem 9.5 is also as strong as possible: If the configuration has cardinality three, CONFIGURATION CONTAINMENT becomes W[1]-hard.

Theorem 9.6. *The* CONFIGURATION CONTAINMENT *problem parameterized by* $t(\Phi) = |X(\Phi)|$ *is* W[1]*-hard, even if* $|\mathcal{P}| = 3$ *and* $|\Phi| = 3$.

Proof. We reduce from the W[1]-complete SEGREGATED PERMUTATION PATTERN MATCHING problem, introduced in Chapter 4, Theorem 4.7.

SEGREGATED PERMUTATION PATTERN MATCHING (SPPM)

Instance:	A permutation σ (the text) of length n, a permutation π (the pattern) of length $k \leq n$ and two positive integers $p \in [k]$, $t \in [n]$.
Parameter:	k
Question:	Is there a matching M of P into T such that $M(i) \leq t$ if and only if $i \leq p$?

Let $C = [n+2]$ and $X(\Phi) = [k+2]$. We define the configuration (ϕ_1, ϕ_2, ϕ_3) as follows:

$$1 > 2 > \cdots > p > k+1 > k+2 > p+1 > \cdots > k \qquad (\phi_1)$$
$$1 > 2 > \cdots > p > k+2 > k+1 > p+1 > \cdots > k \qquad (\phi_2)$$
$$\pi(1) > \cdots > \pi(p) > k+1 > k+2 > \pi(p+1) > \cdots > \pi(k) \qquad (\phi_3)$$

The profile (V_1, V_2, V_3) is defined as follows:

$$1 \succ 2 \succ \cdots \succ t \succ n+1 \succ n+2 \succ t+1 \succ \cdots \succ n \qquad (V_1)$$
$$1 \succ 2 \succ \cdots \succ t \succ n+2 \succ n+1 \succ t+1 \succ \cdots \succ n \qquad (V_2)$$
$$\sigma(1) \succ \cdots \succ \sigma(t) \succ n+1 \succ n+2 \succ \sigma(t+1) \succ \cdots \succ \sigma(n) \qquad (V_3)$$

We claim that the configuration (ϕ_1, ϕ_2, ϕ_3) is contained in the profile (V_1, V_2, V_3) if and only if there is a matching M of π to σ such that $M(i) \leq t$ if and only if $i \leq p$.

"$\rightarrow$" The crucial observation here is that only V_1 and V_2 are possible models for ϕ_1 and ϕ_2 and thus $V_3 \models_\xi \phi_3$ for some function $\xi : [k+2] \rightarrow [n+2]$. Observe that ξ is monotone and $\xi(k+1) = n+1$ and $\xi(k+2) = n+2$. As a consequence, ξ restricted to $[k]$ is a matching of π into σ such that $M(i) \leq t$ if and only if $i \leq p$.

"$\leftarrow$" Here, for $\xi = M \cup \{k+1 \mapsto n+1, k+2 \mapsto n+2\}$, it is easy to verify to $V_1 \models_\xi \phi_1$, $V_2 \models_\xi \phi_2$ and $V_3 \models_\xi \phi_3$. □

As a consequence of these results, we know that a substantial improvement of the algorithm for detecting configurations (Proposition 7.1) is not possible and, in particular, a universally applicable fpt algorithm does not exist unless $\mathsf{FPT} = \mathsf{W[1]}$.

9.3 Summary

In this section we have applied results from permutation pattern matching to the field of domain restrictions. We have obtained combinatorial results (Theorem 9.3) about the number of profiles avoiding $(2, k)$-configurations using the Marcus–Tardos theorem. In addition, we have presented computational results establishing algorithmic bounds for efficient detection of configurations (Theorem 9.4, 9.5, 9.6) making use of a complexity result established in Chapter 4. Our work in this chapter is only preliminary and hopefully further connections between permutation patterns and configurations are to be discovered.

CHAPTER 10

Conclusions and Directions for Future Research

The goal of this thesis is to provide algorithms for detecting structure in permutations and preferences. Structure in permutations is studied in the form of permutation patterns, structure in preferences in the form of domain restrictions. The first part of this thesis contains an extensive complexity analysis of PERMUTATION PATTERN MATCHING (PPM) for generalized patterns (Chapter 4) as well as a novel algorithm for PPM with a runtime of $\mathcal{O}^*(1.79^n)$; the first algorithm to beat the brute-force runtime of $\mathcal{O}^*(2^n)$ (Chapter 5).

The second part concerns structure detection in preferences. First, in Chapter 6, we show for nearly single-peaked preferences that allowing for notions of "nearness" increases the complexity of detecting structure. Then, in Chapter 7, we design approximation and fixed-parameter tractable algorithms to deal with the high complexity of detecting "nearness" to structure. In Chapter 8 we formalize the meaning of single-peakedness in the presence of incomplete information and obtain algorithms for detecting single-peakedness in incomplete preference data. Finally, we observe in Chapter 9 that permutation patterns and domain restrictions are not separate topics but related concepts; their connection is established by relating permutation patterns and forbidden configurations. In addition, we show that certain domain restrictions are very unlikely to appear in random preference data. For a more detailed overview of the results of this thesis, we refer the reader to the summary sections at the end of each chapter.

To sum up, this thesis presents a variety of algorithms for detecting structure in permutations and preferences, applicable in a wide range of scenarios and applications.

The Big Picture

We would now like to identify general lessons that can be learned from this thesis. Let us first consider our results from the perspective of computational complexity. Most algorithmic problems are computationally hard and three reasons for high complexity can be observed.

1. Complexity arises due to unbounded size of the pattern one is looking for. While PPM is NP-complete if the length of the pattern is unbounded [34], it becomes linear-time solvable for bounded length [92]. For generalized permutation patterns the same holds: while NP-complete in the general case (Corollary 4.3), it is polynomial-time solvable for bounded pattern length (Theorem 4.6). Similar results hold for domain restrictions: while the general CONFIGURATION CONTAINMENT problem is NP-complete (Theorem 9.4), it requires only polynomial time to detect a fixed domain restriction that is configuration definable (Proposition 7.1).

2. Computational complexity may also arise due to incomplete information. In Chapter 8 we have seen that even for local weak orders (which possess quite some structure) detecting single-peakedness is NP-hard (Theorem 8.3), although it only requires linear time for profiles of weak orders with a single total order (Theorem 8.6). For permutation patterns we have not considered incompletely specified permutations, although this is an interesting direction for future research.

3. Finally, complexity may arise due to allowing for more flexible notions of structure. In Chapter 6 we have seen that, for example, deleting votes in order to obtain single-peakedness is NP-hard, even though detecting single-peakedness can be done in linear time. Questions of that sort may also be of interest in the context of permutation patterns; for example, asking for the minimal number of elements that have to be removed from a permutation to make it avoid a pattern.

From an algorithmic perspective, this thesis shows how the computational hardness of structure detection can be handled with established techniques in algorithm design: polynomial-time solvable fragments, fixed-parameter algorithms and approximation algorithms. All of these techniques have their strengths and weaknesses and are not applicable in every setting. For example, approximation algorithms are not directly applicable to PPM and it is unclear what a reasonable restricted fragment for detecting nearly single-peakedness would be. However, these techniques taken together deliver excellent tools for solving computationally hard tasks regarding the detection of structure in permutations and preferences.

We would now like to conclude with highlighting several possible research directions that build upon the results presented in this thesis.

Future Directions: Permutation Pattern Matching

Polynomial time algorithms. In Section 4.2.2 we listed several special cases for which PPM is polynomial time solvable. This list, however, is certainly far from being complete. In particular, polynomial time fragments of vincular, bivincular and mesh permutation pattern matching are not known at all.

Other parameters than $k = |P|$. In Section 4.3 we have studied the influence of the length of the pattern on the complexity of the different types of permutation pattern matching problems. Both for generalizations of PPM that are W[1]-hard with respect to k as well as for classical

PPM which is in FPT with respect to k, it is of interest to find out whether other parameters of the input instances lead to fixed parameter tractability results. In Chapter 5 we provided a first result in this vein by designing an algorithm that solves PPM with a worst-case runtime of $\mathcal{O}(1.79^{\mathsf{run}(T)} \cdot n \cdot k)$, where $\mathsf{run}(T)$ denotes the number of alternating runs of T. For future work any permutation statistic (see for instance the list in Appendix A.1 of [105]) could be taken into account for a parameterized complexity analysis of all versions of PPM. An analysis of PPM with respect to several different parameters would then allow us to draw a more detailed picture of the computational landscape of permutation pattern matching.

Patterns in words. In this thesis, we have considered patterns in *permutations*. However, the concept of pattern avoidance respectively containment can easily be extended to patterns in words over ordered alphabets (or permutations on multisets). In a matching of a word W into another word V, copies of the same letter have to be mapped to copies of some letter in the text. The topic of patterns in words has received quite some attention in the last years, see e.g. Heubach and Mansour's monograph *Combinatorics of compositions and words* [98]. The corresponding pattern matching problems have not yet been studied.

Kernelization. Theorem 5.1 shows fixed-parameter tractability of PPM with respect to $\mathsf{run}(T)$. An immediate consequence is that any PPM instance can be reduced by polynomial time preprocessing to an equivalent instance – a kernel – of size depending solely on $\mathsf{run}(T)$. This raises the question whether even a polynomial-sized kernel exists. Such kernels, and in particular polynomial kernels, have been the focus of intensive research in algorithmics [94].

Implementations. At this point, several algorithms exist that solve PPM without imposing restrictions on P and T. The algorithms by Guillemot and Marx [92], Albert et al. [2] and Ahal and Rabinovich [1] seem to be particularly well-suited for small patterns. In contrast, the runtime of our algorithm does not critically depend on $|P|$. Thus, it may be expected that our algorithm is preferable for large patterns. However, only implementations and benchmarks could allow one to settle this question and systematically compare these algorithms.

Future Directions: Structure in Preferences

Nearly Structured Preferences. An obvious direction for future work is to determine the complexity of CANDIDATE PARTITION SINGLE-PEAKED CONSISTENCY. Also, as mentioned in the summary of Chapter 6, the notions of distance to single-peakedness easily carry over to other domain restrictions. The corresponding computational tasks of detecting such structure have not been studied so far, except for the candidate and voter deletion distance [44]. As soon as the complexity of these tasks is settled (which can be expected to be NP-hard in most cases), it is desirable to search for fpt- or approximation algorithms in the spirit of those in Chapter 7.

Structure in Incomplete Preferences. Our work on structure in incomplete preferences, presented in Chapter 8, can be extended in several directions. One direction is to extend our algorithms to notions of nearly single-peakedness. Another direction is the exploration of other domain restrictions such as the single-crossing restriction. Finally, the complexity of WEAK ORDER SINGLE-PEAKED CONSISTENCY remains open; settling this question would be highly desirable.

Connections between Structure in Permutation Patterns and in Preferences. Chapter 9 is only the starting point for a systematic study of the relation of permutation patterns and preferences. So far, we have succeeded in applying the Marcus–Tardos theorem only to domain restrictions containing a configuration of size two. Is it possible to generalize this result to arbitrary domain restrictions? Another research direction is to study the likelihood for specific domain restrictions aiming at (asymptotically) exact results. In particular, the likelihood of domain restrictions under different preference probability distributions would be desirable. Finally, it might be the case that algorithms for permutation pattern detection (such as [1, 2, 92] or the algorithm presented in Chapter 5) yield fast algorithms for detecting domain restrictions; this has yet to be investigated.

Bibliography

[1] S. Ahal and Y. Rabinovich. On complexity of the subpattern problem. *SIAM Journal on Discrete Mathematics*, 22(2):629–649, 2008. (Cited on pages 15, 30, 63, 64, 67, 139, and 140.)

[2] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, and D. Holton. Algorithms for pattern involvement in permutations. In *Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001. (Cited on pages 15, 30, 139, and 140.)

[3] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, H. P. van Ditmarsch, B. D. Handley, C. C. Handley, and J. Opatrny. Longest subsequences in permutations. *Australasian Journal of Combinatorics*, 28:225–238, 2003. (Cited on page 16.)

[4] P. Anand, P. Pattanaik, and C. Puppe, editors. *The Handbook of Rational and Social Choice*. Oxford University Press, 2009. (Cited on page 17.)

[5] D. André. Étude sur les maxima, minima et séquences des permutations. *Annales Scientifiques de l'École Normale Supérieure*, 3(1):121–135, 1884. (Cited on page 39.)

[6] S. Arora and B. Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009. (Cited on page 13.)

[7] K. J. Arrow. A difficulty in the concept of social welfare. *The Journal of Political Economy*, 58(4):328–346, 1950. (Cited on page 17.)

[8] K. J. Arrow. *Social Choice and Individual Values*. John Wiley and Sons, 1951. (Cited on page 17.)

[9] K. J. Arrow, A. Sen, and K. Suzumura, editors. *Handbook of Social Choice and Welfare, Volume 1*. Elsevier, 2002. (Cited on page 17.)

[10] K. J. Arrow, A. Sen, and K. Suzumura, editors. *Handbook of Social Choice and Welfare, Volume 2*. Elsevier, 2010. (Cited on page 17.)

[11] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. (Cited on pages 5 and 122.)

[12] S. Avgustinovich, S. Kitaev, and A. Valyuzhenich. Avoidance of boxed mesh patterns on permutations. *Discrete Applied Mathematics*, 161(1-2):43–51, 2013. (Cited on pages 23 and 28.)

[13] H. Aziz. Testing top monotonicity. *arXiv preprint arXiv:1403.7625*, 2014. (Cited on page 18.)

[14] E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the mahonian statistics. *Séminaire Lotharingien de Combinatoire*, 44:117, 2000. (Cited on pages 23 and 26.)

[15] M. Ballester and G. Haeringer. A characterization of the single-peaked domain. *Social Choice and Welfare*, 36(2):305–322, 2011. (Cited on page 98.)

[16] M. A. Ballester and G. Haeringer. A characterization of the single-peaked domain. *Social Choice and Welfare*, 36(2):305–322, 2011. (Cited on pages 18 and 132.)

[17] S. Barberà, F. Gul, and E. Stacchetti. Generalized median voter schemes and committees. *Journal of Economic Theory*, 61(2):262–289, 1993. (Cited on pages 12 and 18.)

[18] S. Barberà and B. Moreno. Top monotonicity: A common root for single peakedness, single crossing and the median voter result. *Games and Economic Behavior*, 73(2):345–359, 2011. (Cited on page 18.)

[19] S. Barberà, D. Berga, and B. Moreno. Some new domain restrictions in social choice, and their consequences. In *Modeling Decisions for Artificial Intelligence*, volume 8234 of *Lecture Notes in Computer Science*, pages 11–24. Springer, 2013. (Cited on page 18.)

[20] J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989. (Cited on page 17.)

[21] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989. (Cited on page 17.)

[22] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989. (Cited on page 86.)

[23] J. Bartholdi, III and M. Trick. Stable matching with preferences derived from a psychological model. *Operations Research Letters*, 5(4):165–169, 1986. (Cited on pages 18, 114, and 128.)

[24] D. Baumeister, P. Faliszewski, J. Lang, and J. Rothe. Campaigns for lazy voters: Truncated ballots. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 577–584, 2012. (Cited on page 110.)

[25] D. Baumeister, M. Roos, J. Rothe, L. Schend, and L. Xia. The possible winner problem with uncertain weights. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 133–138, Aug. 2012. (Cited on page 19.)

[26] D. Baumeister and J. Rothe. Taking the final step to a full dichotomy of the possible winner problem in pure scoring rules. *Information Processing Letters*, 112(5):186 – 190, 2012. (Cited on pages 19 and 109.)

[27] N. Betzler and B. Dorn. Towards a dichotomy for the possible winner problem in elections based on scoring rules. *Journal of Computer and System Sciences*, 76(8):812–836, 2010. (Cited on pages 19 and 109.)

[28] N. Betzler, S. Hemmann, and R. Niedermeier. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 53–58. AAAI Press, July 2009. (Cited on page 19.)

[29] N. Betzler, A. Slinko, and J. Uhlmann. On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519, 2013. (Cited on page 18.)

[30] D. Black. On the rationale of group decision making. *Journal of Political Economy*, 56(1):23–34, 1948. (Cited on pages 11, 18, and 109.)

[31] M. Bóna. A survey of stack-sorting disciplines. *Electronic Journal of Combinatorics*, 9(2):16, 2003. (Cited on page 1.)

[32] M. Bóna. *Combinatorics of permutations*. Chapman & Hall/CRC, 2004. (Cited on pages 10, 15, and 62.)

[33] C. Borgers. *Mathematics of Social Choice: Voting, Compensation, and Division*. SIAM Society for Industrial and Applied Mathematics, 2009. (Cited on page 17.)

[34] P. Bose, J. F. Buss, and A. Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277 – 283, 1998. (Cited on pages 2, 15, 16, 23, 25, 28, 32, 39, and 138.)

[35] M. Bousquet-Mélou, A. Claesson, M. Dukes, and S. Kitaev. $(2+2)$-free posets, ascent sequences and pattern avoiding permutations. *Journal of Combinatorial Theory, Series A*, 117(7):884–909, 2010. (Cited on pages 23 and 27.)

[36] M. Bouvel and E. Pergola. Posets and permutations in the duplication-loss model: Minimal permutations with descents. *Theoretical Computer Science*, 411(26–28):2487 – 2501, 2010. (Cited on page 1.)

[37] M. Bouvel and D. Rossin. The longest common pattern problem for two permutations. *Pure Mathematics and Applications*, 17(1-2):55–69, 2006. (Cited on page 16.)

[38] M. Bouvel and D. Rossin. A variant of the tandem duplication-random loss model of genome rearrangement. *Theoretical Computer Science*, 410(8–10):847 – 858, 2009. (Cited on page 1.)

[39] M. Bouvel, D. Rossin, and S. Vialette. Longest common separable pattern among permutations. In *Combinatorial Pattern Matching (CPM 2007)*, Lecture Notes in Computer Science, pages 316–327. Springer, 2007. (Cited on page 16.)

[40] P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electronic Journal of Combinatorics*, 18(2):P5, 2011. (Cited on pages 23 and 27.)

[41] F. Brandt, M. Brill, E. Hemaspaandra, and L. A. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 715–722, 2010. (Cited on pages 18, 109, and 127.)

[42] F. Brandt, V. Conitzer, and U. Endriss. Computational social choice. In *Multiagent Systems*. MIT Press, 2013. (Cited on page 17.)

[43] R. Bredereck, 2012. Personal communication. (Cited on page 83.)

[44] R. Bredereck, J. Chen, and G. Woeginger. Are there any nicely structured preference profiles nearby? In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 62–68, 2013. (Cited on pages 19, 83, 95, and 139.)

[45] R. Bredereck, J. Chen, and G. Woeginger. A characterization of the single-crossing domain. *Social Choice and Welfare*, 41(4):989–998, 2013. (Cited on pages 18 and 98.)

[46] M.-L. Bruner and M. Lackner. A fast algorithm for permutation pattern matching based on alternating runs. In *Proceedings of the 13th Scandinavian Workshop on Algorithm Theory (SWAT 2012)*, volume 7357 of *Lecture Notes in Computer Science*, pages 261–270. Springer, 2012. (Cited on page 39.)

[47] M.-L. Bruner and M. Lackner. The computational landscape of permutation patterns. *Pure Mathematics and Applications*, 2014. Accepted for publication. (Cited on page 23.)

[48] M.-L. Bruner and M. Lackner. The likelihood of structure in preference profiles. In *Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPref 2014)*, 2014. Accepted for publication. (Cited on page 129.)

[49] M.-S. Chang and F.-H. Wang. Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. *Information Processing Letters*, 43(6):293–295, 1992. (Cited on page 16.)

[50] K. Chaudhuri, K. Chen, R. Mihaescu, and S. Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–570. ACM, 2006. (Cited on page 1.)

[51] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. (Cited on page 105.)

[52] V. Conitzer. Eliciting single-peaked preferences using comparison queries. *Journal of Artificial Intelligence Research*, 35:161–191, 2009. (Cited on pages 18 and 114.)

[53] V. Conitzer. Making decisions based on the preferences of multiple agents. *Commun. ACM*, 53(3):84–94, Mar. 2010. (Cited on page 16.)

[54] V. Conitzer and T. Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, pages 392–397, 2002. (Cited on page 19.)

[55] C. H. Coombs. *A Theory of Data*. John Wiley & Sons, 1964. (Cited on pages 130 and 132.)

[56] D. Cornaz, L. Galand, and O. Spanjaard. Bounded single-peaked width and proportional representation. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 270–275, 2012. (Cited on page 19.)

[57] D. Cornaz, L. Galand, and O. Spanjaard. Kemeny elections with bounded single-peaked or single-crossing width. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 76–82, 2013. (Cited on page 19.)

[58] M. Crochemore, C. S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In *String Processing and Information Retrieval*, volume 8214 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2013. (Cited on page 16.)

[59] F. N. David and D. E. Barton. *Combinatorial chance*. Griffin London, 1962. (Cited on page 62.)

[60] G. Demange. Single-peaked orders on a tree. *Mathematical Social Sciences*, 3(4):389–396, 1982. (Cited on page 18.)

[61] J.-P. Doignon and J.-C. Falmagne. A polynomial time algorithm for unidimensional unfolding representations. *Journal of Algorithms*, 16(2):218–233, 1994. (Cited on page 114.)

[62] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in AI: An overview. *Artificial Intelligence*, 175(7–8):1037–1052, 2011. (Cited on page 16.)

[63] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. (Cited on pages 14 and 32.)

[64] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer London, 2013. (Cited on page 14.)

[65] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference (WWW 2001)*, pages 613–622. ACM Press, 2001. (Cited on pages 18, 86, 87, and 110.)

[66] S. Elizalde and M. Noy. Consecutive patterns in permutations. *Advances in Applied Mathematics*, 30(1):110–125, 2003. (Cited on pages 23 and 28.)

[67] E. Elkind, P. Faliszewski, and A. Slinko. Clone structures in voters' preferences. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC 2012)*, pages 496–513, 2012. (Cited on pages 18 and 19.)

[68] E. Elkind and M. Lackner. On detecting nearly structured preference profiles. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI Press, 2014. Accepted for publication. (Cited on page 95.)

[69] E. Ephrati and J. Rosenschein. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1–4):13–67, 1997. (Cited on page 18.)

[70] G. Erdélyi, M. Lackner, and A. Pfandler. Computational aspects of nearly single-peaked electorates. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013)*. AAAI Press, 2013. (Cited on page 71.)

[71] B. Escoffier, J. Lang, and M. Öztürk. Single-peaked consistency and its complexity. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 366–370, 2008. (Cited on pages 18, 19, 72, 73, 88, 90, 109, 110, and 114.)

[72] S. Even and R. Bar-Yehuda. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981. (Cited on pages 6 and 104.)

[73] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006. (Cited on page 110.)

[74] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003. (Cited on page 110.)

[75] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. The complexity of manipulative attacks in nearly single-peaked electorates. *Artificial Intelligence*, 207:69–99, 2014. (Cited on pages 18, 19, 72, 75, and 107.)

[76] P. Faliszewski, E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Information and Computation*, 209(2):89–107, 2011. (Cited on pages 18 and 109.)

[77] H. Fernau. Parameterized algorithms for d-hitting set: The weighted case. *Theoretical Computer Science*, 411(16–18):1698 – 1713, 2010. (Cited on page 105.)

[78] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009. (Cited on page 62.)

[79] J. Flum and M. Grohe. Model-checking problems as a basis for parameterized intractability. *Logical Methods in Computer Science*, 1(1), 2005. (Cited on page 31.)

[80] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 2006. (Cited on pages 14 and 107.)

[81] F. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer, 2010. (Cited on page 39.)

[82] M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 1997. (Cited on page 85.)

[83] J. Fürnkranz and E. Hüllermeier, editors. *Preference learning*. Springer, 2011. (Cited on page 16.)

[84] Z. Füredi and P. Hajnal. Davenport-Schinzel theory of matrices. *Discrete Mathematics*, 103(3):233 – 251, 1992. (Cited on page 15.)

[85] W. Gaertner. *A Primer in Social Choice Theory: Revised Edition*. Oxford University Press, 2009. (Cited on page 17.)

[86] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. (Cited on pages 84 and 115.)

[87] P. Gawrychowski and P. Uznanski. Order-preserving pattern matching with k mismatches. *arXiv preprint arXiv:1309.6453*, 2013. (Cited on page 16.)

[88] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: The anatomy of recommender systems. In *Proceedings of the 3rd Annual Conference on Autonomous Agents (AGENTS 1999)*, pages 434–435. ACM Press, 1999. (Cited on page 18.)

[89] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41(4):587–601, 1973. (Cited on page 17.)

[90] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. (Cited on page 13.)

[91] J. Goldsmith and U. Junker, editors. *Preferences*, volume 29(4). AI Magazine, 2008. (Cited on page 16.)

[92] S. Guillemot and D. Marx. Finding small patterns in permutations in linear time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 82–101, 2014. (Cited on pages 2, 15, 25, 31, 37, 134, 138, 139, and 140.)

[93] S. Guillemot and S. Vialette. Pattern matching for 321-avoiding permutations. In *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 1064–1073. Springer, 2009. (Cited on page 16.)

[94] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. (Cited on page 139.)

[95] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. (Cited on page 106.)

[96] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997. (Cited on page 109.)

[97] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005. (Cited on page 109.)

[98] S. Heubach and T. Mansour. *Combinatorics of compositions and words*. Chapman & Hall/CRC, 2009. (Cited on page 139.)

[99] L. Ibarra. Finding pattern matchings for permutations. *Information Processing Letters*, 61(6):293–295, 1997. (Cited on page 16.)

[100] K. Inada. A note on the simple majority decision rule. *Econometrica*, 32(4):525–531, 1964. (Cited on page 18.)

[101] K. Inada. The simple majority decision rule. *Econometrica*, 37(3):490–506, 1969. (Cited on page 18.)

[102] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):pp. 81–93, 1938. (Cited on page 74.)

[103] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335 – 349, 2008. (Cited on page 104.)

[104] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama. Order preserving matching. *arXiv preprint arXiv:1302.4064*, 2013. (Cited on page 16.)

[105] S. Kitaev. *Patterns in Permutations and Words*. Springer, 2011. (Cited on pages 15, 24, 26, 28, 132, and 139.)

[106] M. Klazar. The Füredi-Hajnal conjecture implies the Stanley-Wilf conjecture. In *Formal power series and algebraic combinatorics (FPSAC 2000)*, pages 250–255. Springer, 2000. (Cited on page 15.)

[107] A. Klenke. *Probability theory: a comprehensive course*. Springer, 2008. (Cited on page 62.)

[108] V. Knoblauch. Recognizing one-dimensional Euclidean preference profiles. *Journal of Mathematical Economics*, 46(1):1 – 5, 2010. (Cited on pages 130 and 132.)

[109] D. E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968. (Cited on pages 15, 24, and 62.)

[110] M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, and T. Waleń. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, 113(12):430 – 433, 2013. (Cited on pages 16 and 30.)

[111] M. Lackner. Incomplete preferences in single-peaked electorates. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI Press, 2014. Accepted for publication. (Cited on page 109.)

[112] J. Lang, M. Pini, F. Rossi, D. Salvagnin, K. Venable, and T. Walsh. Winner determination in voting trees with incomplete preferences and weighted votes. *Autonomous Agents and Multiagent Systems*, 25(1):130–157, 2012. (Cited on page 19.)

[113] H. Levene and J. Wolfowitz. The covariance matrix of runs up and down. *The Annals of Mathematical Statistics*, 15(1):58–69, 1944. (Cited on page 62.)

[114] E. Mäkinen. On the longest upsequence problem for permutations. *International Journal of Computer Mathematics*, 77(1):45–53, 2001. (Cited on page 16.)

[115] A. Marcus and G. Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153–160, 2004. (Cited on pages 15, 130, and 132.)

[116] N. Mattei, J. Forshee, and J. Goldsmith. An empirical study of voting rules and manipulation with large datasets. In *Proceedings of the 4th International Workshop on Computational Social Choice (COMSOC 2012)*, 2012. (Cited on pages 71 and 129.)

[117] N. Mattei and T. Walsh. Preflib: A library of preference data. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*. Springer, 2013. (Cited on page 109.)

[118] I. McLean and A. Urken. *Classics of Social Choice*. University of Michigan Press, 1995. (Cited on page 17.)

[119] J. Mirrlees. An exploration in the theory of optimal income taxation. *Review of Economic Studies*, 38:175–208, 1971. (Cited on page 18.)

[120] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. (Cited on page 14.)

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

[122] C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003. (Cited on page 13.)

[123] D. Pennock, E. Horvitz, and C. Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, pages 729–734. AAAI Press, 2000. (Cited on page 18.)

[124] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Incompleteness and incomparability in preference aggregation: Complexity results. *Artificial Intelligence*, 175(7-8):1272 – 1289, 2011. (Cited on pages 19 and 109.)

[125] K. W. Roberts. Voting over income tax schedules. *Journal of Public Economics*, 8(3):329–340, 1977. (Cited on page 12.)

[126] F. Rossi, K. B. Venable, and T. Walsh. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan and Claypool, 2011. (Cited on page 16.)

[127] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975. (Cited on page 17.)

[128] S. Saxena and V. Yugandhar. Parallel algorithms for separable permutations. *Discrete Applied Mathematics*, 146(3):343–364, 2005. (Cited on page 16.)

[129] C. Schensted. Longest increasing and decreasing subsequences. *Classic Papers in Combinatorics*, pages 299–311, 1987. (Cited on pages 16 and 92.)

[130] A. Sen. A possibility theorem on majority decisions. *Econometrica*, 34:491–499, 1966. (Cited on page 18.)

[131] R. Simion and F. W. Schmidt. Restricted permutations. *European Journal of Combinatorics*, 6:383–406, 1985. (Cited on pages 15 and 24.)

[132] P. Skowron, L. Yu, P. Faliszewski, and E. Elkind. The complexity of fully proportional representation for single-crossing electorates. In *Proceedings of the 6th International Symposium on Algorithmic Game Theory (SAGT 2013)*, pages 1–12, 2013. (Cited on page 18.)

[133] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Transactions on Database Systems*, 36(3):19:1–19:45, Aug. 2011. (Cited on page 18.)

[134] E. Steingrímsson. Generalized permutation patterns - a short survey. *LMS Lecture Note Series*, 376:137–152, 2010. (Cited on page 26.)

[135] X. Sui, A. Francois-Nienaber, and C. Boutilier. Multi-dimensional single-peaked consistency and its approximations. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2013. (Cited on pages 18, 71, and 129.)

[136] M. Trick. Recognizing single-peaked preferences on a tree. *Mathematical Social Sciences*, 17(3):329–334, 1989. (Cited on page 18.)

[137] M. Wahlström. *Algorithms, measures, and upper bounds for satisfiability and related problems*. PhD thesis, Linköping University, 2007. (Cited on page 105.)

[138] T. Walsh. Uncertainty in preference elicitation and aggregation. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, pages 3–8. AAAI Press, July 2007. (Cited on pages 19 and 109.)

[139] T. Walsh. Complexity issues in preference elicitation and manipulation. In *Proceedings of The 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM 2008)*, Jan. 2008. (Cited on page 19.)

[140] T. Walsh. Complexity of terminating preference elicitation. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 967–974. International Foundation for Autonomous Agents and Multiagent Systems, May 2008. (Cited on page 19.)

[141] J. West. *Permutations with forbidden subsequences, and, stack-sortable permutations*. PhD thesis, Massachusetts Institute of Technology, 1990. (Cited on page 1.)

[142] L. Xia and V. Conitzer. Determining possible and necessary winners under common voting rules given partial orders. *Journal of Artificial Intelligence Research*, 41(2):25–67, 2011. (Cited on pages 19 and 109.)

[143] L. Yu, H. Chan, and E. Elkind. Multiwinner elections under preferences that are single-peaked on a tree. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 425–431, 2013. (Cited on page 18.)

[144] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 681–690, 2006. (Cited on page 106.)

Curriculum Vitae

Martin Lackner

CONTACT	Vienna University of Technology Institute of Information Systems Database and Artificial Intelligence Group
ADDRESS	Favoritenstraße 9-11 1040 Vienna, Austria
EMAIL	lackner@dbai.tuwien.ac.at
WEB	http://www.dbai.tuwien.ac.at/staff/lackner/

Education

2010–2014	Vienna University of Technology Doctoral studies in the program “Mathematical Logic in Computer Science”
2009	University of Illinois at Urbana-Champaign, USA Semester Abroad Focus on mathematical logic, artificial intelligence, computational complexity
2005–2010	Vienna University of Technology Diploma program: Technical Mathematics Area of concentration: Mathematics in Computer Science Degree: Dipl.Ing.

Publications

2014 Martin Lackner. Incomplete preferences in single-peaked electorates. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

Edith Elkind and Martin Lackner. On detecting nearly structured preference profiles. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

Martin Kronegger, Martin Lackner, Andreas Pfandler and Reinhard Pichler. A parameterized complexity analysis of generalized CP-nets. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2014. Accepted for publication.

Marie-Louise Bruner and Martin Lackner. The computational landscape of permutation patterns. *Pure and Applied Mathematics*, 2014. Accepted for publication.

Marie-Louise Bruner and Martin Lackner. The likelihood of structure in preference profiles. In *Proceedings of the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPref 2014)*, 2014. Accepted for publication.

2013 Gabór Erdélyi, Martin Lackner, and Andreas Pfandler. Computational aspects of nearly single-peaked electorates. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2013)*, 2013.

Martin Kronegger, Martin Lackner, Andreas Pfandler, and Reinhard Pichler. A parameterized complexity analysis of generalized CP-nets. In *Proceedings of the 7th Multidisciplinary Workshop on Advances in Preference Handling (MPref 2013)*, 2013.

Martin Lackner. Incomplete preferences in single-peaked electorates. In *Proceedings of the 7th Multidisciplinary Workshop on Advances in Preference Handling (MPref 2013)*, 2013.

2012 Marie-Louise Bruner and Martin Lackner. A fast algorithm for permutation pattern matching based on alternating runs. In *Proceedings of the 13th Scandinavian Workshop on Algorithm Theory (SWAT 2012)*, volume 7357 of *Lecture Notes in Computer Science*, pages 261–270. Springer, 2012.

Martin Lackner and Andreas Pfandler. Fixed-parameter algorithms for finding minimal models. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 85–95. AAAI Press, 2012.

Martin Lackner and Andreas Pfandler. Fixed-parameter algorithms for closed world reasoning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 492–497. IOS Press, 2012.

Martin Lackner, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut on graphs of bounded clique-width. In *Proceedings of the 6th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2012)*, volume 7402 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2012.

Leo Brueggeman, Michael R. Fellows, Rudolf Fleischer, Martin Lackner, Christian Komusiewicz, Yiannis Koutis, Andreas Pfandler, and Frances A. Rosamond. Train marshalling is fixed parameter tractable. In *Proceedings of the Sixth International Conference on Fun with Algorithms (FUN 2012)*, volume 7288 of Lecture Notes in Computer Science, pages 51-56. Springer, 2012.

Gabór Erdélyi, Martin Lackner, and Andreas Pfandler. The complexity of nearly single-peaked consistency. In *Proceedings of the 4th International Workshop on Computational Social Choice (COMSOC 2012)*, 2012.

Marie-Louise Bruner and Martin Lackner. From peaks to valleys, running up and down: Fast permutation pattern matching. *TinyToCS*, 2012.