CHAPTER 11

# Having a Hard Time? Explore Parameterized Complexity!

Britta Dorn and Ildikó Schlotter

## 11.1 Motivation

More often than not, life teems with difficult problems. This is not less true if you happen to be a researcher in computational social choice; however, in this case you can spend considerable time focusing only on *computational* hardness.

Collective decision making has been studied from various aspects. Political science, economics, mathematics, logic, and philosophy have all contributed to the area of social choice. With the advance of computer science, computational issues have become more and more important. Taking a casual look at the landscape of computational problems in social choice, we find an abundance of hard problems. Within the theory of voting, already winner determination is NP-hard for several voting rules like Dodgson, Young, or Kemeny voting. Considering certain forms of manipulation, control, or bribery in elections, or dealing with partial information results in computationally hard problems as well. We can find examples in every area of social choice, let it be judgment aggregation, fair division of goods, or matching under preferences.

**Computational complexity: the classical approach.** When considering the computational tractability of a given problem, we focus on the time and space necessary for an algorithm to solve it. In most cases, however, space is not the scarcest resource, and therefore whether an algorithm is considered tractable or not depends on its running time. Of course, running times depend on the actual input, and to overcome this rather cumbersome difficulty, classical complexity theory teaches us to view the running time of an algorithm as a function of the *length* of its input. More precisely, the running time $T(n)$ of a given algorithm $\mathcal{A}$ is defined as the maximum number of computational steps performed by $\mathcal{A}$ on any input of length $n$. Using this notion, a broadly accepted rule of thumb is to consider $\mathcal{A}$ (and the problem solved by $\mathcal{A}$) tractable if $T(n)$ is a *polynomial* of $n$.

To grasp the notion of *computational intractability*, classical complexity theory offers a hierarchy of complexity classes, but here we only focus on the central concept of NP-hardness. Instead of repeating the formal definition here, we only would like to recall its most vital property. Namely, there is strong evidence

indicating that NP-hard problems are *not* solvable in polynomial time. From a practical point of view, this means that we cannot expect to find an algorithm solving an NP-hard problem that runs in reasonable time for large inputs.

Over the years, researchers facing NP-hard problems have come up with numerous strategies to deal with intractability. Sometimes focusing on easy special cases can be enough. In many areas, approximation algorithms turned out to be extremely useful. Randomization and parallel computing might also help us reduce the running time, especially when combined with other approaches. Lately, ever-growing computational capacities have made exponential-time (exact) algorithms a viable choice in some cases. And when theory does not seem to offer any help, heuristics still play an important role.

All of these strategies might be useful in computational social choice too. However, there is one crucial aspect shared by these approaches which dooms them inefficient in a certain way: they are all *one-dimensional* in the sense that they regard the running time merely as a function of the input length. In reality, there are several properties of the input, explicit or implicit, that heavily influence the complexity of the problem, and to neglect these is a deep source of inefficiency.

**Parameterized complexity.** So far the only well-developed framework that uses a multidimensional approach to deal with computationally hard problems is *parameterized complexity*. This approach, developed first by Downey and Fellows (1999), considers the complexity of a given problem with respect to several so-called *parameters*, and views the running time of a given algorithm as the function of both the input length and the parameters. This simple idea allows us to draw a much more detailed map of the complexity of a problem.

Each instance of a parameterized problem $P$ is a pair $(\mathcal{I}, k)$ consisting of an input $\mathcal{I}$ and a parameter $k$, which is usually an integer (we will explain later how to handle multiple parameters within this framework). Since we are mostly dealing with NP-hard problems, we cannot expect a polynomial-time algorithm for $P$. Instead, what we are interested in is whether the exponential explosion in the running time can be, in a sense, attributed to the parameter. More precisely, we ask if $P$ admits an algorithm that, on an instance $(\mathcal{I}, k)$ runs in time

$$f(k) \cdot |\mathcal{I}|^{O(1)}$$

for some computable function $f$. Such an algorithm is called *fixed-parameter tractable* (FPT), and the class of parameterized problems solvable by an FPT algorithm is denoted FPT. Usually, the function $f$ is exponential (or worse), but observe that the dependency of the running time on the input length $|\mathcal{I}|$ is a polynomial of constant degree. Hence the essential property of an FPT algorithm: it works fast whenever the parameter value $k$ is a small integer. Intuitively, this indicates that the source of the computational hardness of $P$ is the parameter: if $k$ is small, our instance is tractable, but as $k$ grows, it quickly becomes intractable.

This approach has great potential from a practical perspective: if some parameter is likely to be small in typical real-world instances, then an FPT algorithm can be highly efficient in practice. We can examine the computational complexity of our problem from many different aspects by choosing different parameters and

searching for FPT algorithms with each parameterization—we hence exploit the structure of the problem that is given in the input.

**Why use parameterized complexity in social choice?** Apart from the general advantages of the parameterized framework, there are two additional reasons why it might be particularly helpful in the field of computational social choice.

First, a typical problem in collective decision making contains a handful of natural parameters that, in certain realistic scenarios, are likely to have small values. The most obvious examples are the number of agents or alternatives present, but for a typical problem we can easily detect several natural possibilities for parameterization that may lead to efficient FPT algorithms. This phenomenon can be explained by the fact that most problems in social choice model some real-world situation, and such models tend to have a composite nature, involving various entities and relations between them. Examples include the amount of variety in a voting profile, the budget in a bribery scenario, or the 'distance' from an instance with a certain desirable property, such as single-peakedness for voting profiles, stability for a matching, or envy-freeness of an allocation.

Second, certain problems in the area of social choice have the curious property that their computational hardness might be, in fact, desirable. Such situations often arise when the computational problem models actions of a malicious agent; to name some examples, we can think about bribery, manipulation, or control of some decision making process. For such a problem, computational hardness means that the given process (e.g., a voting rule) is *safe* in the sense that a malicious agent necessarily faces a computationally intractable situation.

Note however, that simple NP-hardness might not prevent malicious acts in reality: as we have argued earlier, even NP-hard problems might admit efficient algorithms that are applicable in practice. Thus, in such cases a more detailed complexity analysis can become crucial—and this is exactly what we can accomplish by studying our problem from the parameterized aspect. Using the intractability theory of the parameterized framework (see Section 11.3), we can provide evidence that certain problems are not fixed-parameter tractable.

Parameterized complexity can hence contribute to a better evaluation of the hardness of the problem in two ways: on the one hand, fixed-parameter tractability with respect to a parameter shows that NP-hardness might only constitute a theoretical barrier, in particular in applications where the value of this parameter is small. On the other hand, parameterized complexity theory may help to justify the shield provided by computational complexity: if a problem belongs to one of the parameterized hardness classes with respect to a parameter $k$, it is unlikely that an efficient algorithm can be found to solve it, even for small values of $k$.

**Relation to existing literature and goal of this chapter.** In the last decade, parameterized complexity has been applied with great success to many problems in computational social choice. We refer to several surveys overviewing this process, starting with the work by Lindner and Rothe (2008), followed by the work of Betzler et al. (2012) on voting problems, the article by Bredereck et al. (2014) presenting challenges in parameterized algorithmics for computational so-

cial choice, and the recent article by Faliszewski and Niedermeier (2015). The goal of this chapter is not to add another survey of current results, trends and challenges, but to provide a comprehensive introduction for anyone interested to get into to this attractive area of research, tailored to applicability in computational social choice, and illustrated with helpful examples.

The classical reference on parameterized complexity is the book by Downey and Fellows (1999), see also the new edition (Downey and Fellows, 2013). The emphasis in the book by Flum and Grohe (2006) is on complexity, and in the book by Niedermeier (2006) on algorithmic techniques. For the most recent advances in parameterized algorithmic techniques, we refer to the book by Cygan et al. (2015).

**Organization.** We will first present in Section 11.2 some of the basic algorithmic techniques for obtaining fixed-parameter tractability results, such as bounded search trees, data reduction and problem kernels, integer linear programming, and color-coding. We will also explain how to handle multiple parameters. We then turn to parameterized intractability in Section 11.3 where we deal with FPT reductions and the most common parameterized complexity classes. Some more advanced techniques like lower bounds for kernelization and the relation between approximation and parameterized algorithms are presented in Section 11.4. We finish with our conclusions in Section 11.5.

## 11.2 Basic Algorithmic Techniques

To illustrate some basic techniques for designing FPT algorithms, we will use the classical VERTEX COVER problem. Given a graph $G$, a *vertex cover* is a set $S$ of vertices in $G$ such that each edge of $G$ has at least one endpoint in $S$.

| VERTEX COVER: | |
|---|---|
| Input: | An undirected graph $G$, and an integer $k$. |
| Question: | Does $G$ contain a vertex cover of size at most $k$? |

Although this problem itself is not about collective decision making, we believe that its importance as a graph problem renders VERTEX COVER essential also to the researchers of this area. VERTEX COVER is a graph problem belonging to the 21 problems proved to be NP-complete by Karp in his seminal paper (Karp, 1972). Thus, we obviously cannot hope to solve this problem by a polynomial-time algorithm. Given the central role of VERTEX COVER in graph theory, several researchers have attempted to design algorithms for it that would perform well in practical situations. In recent decades, VERTEX COVER became one of the most prominent problems in parameterized complexity, showing how successfully this framework can be applied in practice.

**Brute force approach.** Let $(G, k)$ be an instance of VERTEX COVER with $G$ having $n$ vertices. The most simple, brute force approach is the following: try every possible set $S$ of at most $k$ vertices, and check if $S$ is indeed a vertex cover. Since

this latter condition for a given set $S$ can be checked in $O(|E(G)|)$ time, the whole process can be performed in $\binom{n}{k}O(|E(G)|)$ time.[1]

Clearly, we can assume that $G$ is a simple graph, and we may also assume $|E(G)| \leqslant k(n-1) = O(nk)$: since $k$ vertices can *cover* (i.e., be adjacent to) at most $k(n-1)$ edges, $|E(G)| > k(n-1)$ would immediately prove $(G,k)$ to be a 'no'-instance. Using this, the brute force algorithm described above has running time $\binom{n}{k}O(nk) = O(kn^{k+1})$, which becomes intractable already for relatively small graphs: it cannot even deal with an instance where $n = 100$ and $k = 10$.

In what follows, we shall see some basic techniques in parameterized complexity that can be used to design much more efficient algorithms. Currently the fastest algorithm for VERTEX COVER, developed by Chen et al. (2010), runs in time $O(1.2738^k + kn)$. This renders VERTEX COVER solvable even for instances as large as $n = 10^6$ and $k = 40$.

### 11.2.1 Bounded Search Tree

Let us start with a simple observation that allows us to create a more efficient algorithm for VERTEX COVER: if $S$ is a vertex cover for $G$, then for any edge $e$ of $G$, at least one of its endpoints must belong to $S$. The basic idea is to *'guess'* which endpoint of $e$ belongs to $S$. Of course, 'guessing' means that we have to check both possible outcomes of such a guess, which can be thought of as creating a *branching* in our algorithm. The key to the efficiency of such an approach is the following: if $S$ contains at most $k$ vertices, then we need to perform at most $k$ such guesses, resulting in at most $2^k$ possibilities in total.

Before elaborating these ideas in a more general form, let us discuss in detail how this approach works for VERTEX COVER.

#### Example: Bounded Search Tree for VERTEX COVER

Let us be given an instance $(G,k)$ of VERTEX COVER. Our algorithm starts with an empty set $S$, and adds vertices to $S$ one by one to create a vertex cover. The general step is to pick an edge $e = \{u,v\} \in E(G)$ that is not yet covered by $S$, and guess which endpoint of $e$ should be put into $S$. In other words, the algorithm performs a branching into two directions, adding $u$ to $S$ in the one branch, and adding $v$ to $S$ in the other. Then the algorithm proceeds recursively in both branches, decreasing the parameter $k$ to $k-1$ in both branches. The algorithm stops if either all edges are covered by $S$ in which case it outputs $S$ as a solution, or if the parameter reaches 0 in which case it stops without producing a solution. If no solution is found in any of the branches, then the algorithm returns 'no'.

Let VC-BST$(G,k,S)$ denote a call for the above algorithm with input graph $G$, parameter $k$ and a set $S \subseteq V(G)$ which is the partial solution found so far; see Algorithm 1 for a more formal description.

[1] Here and later on, we will rely on the standard notation in graph theory, as used for example in the book by Diestel (2005). In particular, $V(G)$ denotes the set of vertices of $G$, and $E(G)$ denotes the set of edges of $G$.

**Algorithm 1** Search tree algorithm for VERTEX COVER

```
procedure VC-BST(G, k, S)
  if there exists an edge {u, v} ∈ E(G) with {u, v} ∩ S = ∅ then
    if k > 0 then
      Branch 1: VC-BST(G, k − 1, S ∪ {u});
      Branch 2: VC-BST(G, k − 1, S ∪ {v});
    else output 'no';
  else output S;
```

Algorithms with a recursive structure that use branchings similarly as in VC-BST are called *search tree algorithms*. A useful representation is to think of each call of the given algorithm as a node in a rooted tree $T$, where the children of a node are the recursive calls performed in the given call as a result of branching.

The expression *bounded search tree* refers to the fact that to obtain an efficient algorithm, we need to bound the size of $T$ (that is, $|V(T)|$). If $F(|\mathcal{I}|, k)$ is an upper bound on the time necessary for the computations in any given node of the search tree (where $\mathcal{I}$ and $k$ are the input and the parameter values provided for the initial call), then the running time of the whole search tree algorithm is at most $F(|\mathcal{I}|, k) \cdot |V(T)|$. Hence, if both the size of the search tree and $F(|\mathcal{I}|, k)$ are fixed-parameter tractable, then the resulting running time is also FPT. In a typical scenario, $F(|\mathcal{I}|, k)$ is simply a polynomial in $|\mathcal{I}|$, and the size of the search tree is bounded by a function of the parameter $k$. The bound on $|V(T)|$ is often achieved by providing a limit both on the maximum number of branches, say $b$, and the depth of the search tree, say $d$, implying $|V(T)| \leqslant \sum_{i=0}^{d} b^i = O(b^{d+1})$.

In our example for VERTEX COVER, the size of the search tree associated with a run of Algorithm VC-BST$(G, k, \emptyset)$ is at most $2^{k+1} - 1$. Since the computation in each node takes time $O(|E(G)|)$, we obtain a running time of the form $O(2^k|E(G)|)$. This shows that VERTEX COVER is FPT with respect to parameter $k$.

### Example: Bounded Search Tree for MINIMAL APPROVAL VOTING

MINIMAX APPROVAL VOTING models a situation in voting where we aim to find a committee of pre-defined size that minimizes the maximum distance between any vote and the given committee. Formally, an *approval election* is a pair $(\mathcal{C}, \mathcal{V})$ where $\mathcal{C}$ is a set of *candidates* and $\mathcal{V}$ is a collection of *votes*. Each vote is a subset of the candidates approved by the given voter. We call a subset $C \subseteq \mathcal{C}$ a *committee*, and we define the distance of a vote $v$ and the committee $C$ as their symmetric difference $\text{dist}(C, v) = |C \setminus v| + |v \setminus C|$.

MINIMAX APPROVAL VOTING :
Input: An approval election $\mathcal{E} = (\mathcal{C}, \mathcal{V})$, integers $k$ and $d$.
Question: Does there exist a committee $C \subseteq \mathcal{C}$ with $|C| = k$ such that $\text{dist}(C, v) \leqslant d$ for any $v \in \mathcal{V}$?

Let us present a bounded search tree algorithm for MINIMAX APPROVAL VOTING proposed by Misra et al. (2015) that is fixed-parameter tractable with respect to

the parameter $d$ (see also Cygan et al. (2016) for a note on the running time).

The algorithm starts from an appropriate candidate committee $C_0$ of size $k$, and tries to find a fixed solution $S$ by iteratively modifying $C_0$. Initially, we take any vote $v_0 \in \mathcal{V}$, and either add or delete at most $d$ candidates from it to obtain a committee $C_0$ of size $k$ (if this is not possible, then $v_0$ cannot be at distance at most $d$ from *any* size-$k$ committee, so we can output 'no'). By the triangle inequality, we get $\mathrm{dist}(C_0, S) \leqslant \mathrm{dist}(C_0, v_0) + \mathrm{dist}(v_0, S) \leqslant 2d$.

The algorithm then calls a recursive procedure MAV-BST$(C, \delta)$ that keeps track of our candidate committee $C$ and an upper bound $\delta$ on $\mathrm{dist}(C, S)$, initially set to $C_0$ and $2d$, respectively; see Algorithm 2 for a description.

**Algorithm 2** Search tree algorithm for MINIMAX APPROVAL VOTING

**procedure** MAV-BST$(C, \delta)$
  **if** $\delta < 0$ **then** output 'no';
  **else if** $\mathrm{dist}(C, v) > d + \delta$ for some $v \in \mathcal{V}$ **then** output 'no';
  **else if** $\mathrm{dist}(C, v) \leqslant d$ for each $v \in \mathcal{V}$ **then** output $C$;
  **else**
    choose $v \in \mathcal{V}$ such that $\mathrm{dist}(C, v) > d$;
    **if** $|v \setminus C| \leqslant d + 1$ **then** $P_1 \leftarrow v \setminus C$;
    **else** fix any $P_1 \subseteq v \setminus C$ with $|P_1| = d + 1$;
    **if** $|C \setminus v| \leqslant d + 1$ **then** $P_2 \leftarrow C \setminus v$;
    **else** fix any $P_2 \subseteq C \setminus v$ with $|P_2| = d + 1$;
    **for all** $c_1 \in P_1$ and $c_2 \in P_2$ **do**
      Branch $(c_1, c_2)$: MAV-BST$(C \cup \{c_1\} \setminus \{c_2\}, \delta - 2)$;

At each step, MAV-BST first checks certain simple stopping conditions: assuming $\mathrm{dist}(C, S) \leqslant \delta$, neither $\delta < 0$ nor $\mathrm{dist}(C, v) > d + \delta$ for some $v \in \mathcal{V}$ can hold (the latter follows from $\mathrm{dist}(C, v) \leqslant \mathrm{dist}(C, S) + \mathrm{dist}(S, v)$). So if one of these conditions holds, then the algorithm returns 'no' correctly. Otherwise, MAV-BST searches for a vote $v \in \mathcal{V}$ whose distance from $C$ is more than $d$. If no such vote exists, then it outputs $C$ as a solution; otherwise, $\mathrm{dist}(C, v) > d$ implies that $S$ must be 'closer' to $v$ than $C$. The algorithm tries to decrease the distance of $C$ from $v$ by adding a candidate $c_1 \in v \setminus C$ to $C$ and deleting a candidate $c_2 \in C \setminus v$ from $C$; note that this way the size of the committee remains $k$. In fact, by $|v \setminus S| \leqslant d$, any subset $P_1$ of $v \setminus C$ of size $d + 1$ must contain a candidate in $S$. Similarly, we can use any subset $P_2$ of $C \setminus v$ of size $d + 1$ instead of $C \setminus v$. Hence, for some $c_1 \in P_1$ and $c_2 \in P_2$, branch $(c_1, c_2)$ is correct in the sense that $c_1 \in S$ and $c_2 \notin S$.

To analyze the running time of MAV-BST, let us calculate the size of the search tree. At each branching step, there are at most $(d + 1)^2$ ways to choose $c_1$ and $c_2$. Let us give an upper bound on the depth of the search tree: initially, we have $\delta \leqslant 2d$, and we decrease $\delta$ by 2 with each recursion, stopping whenever it becomes negative. Hence, the depth of the search tree is at most $d$, and thus contains at most $((d + 1)^2)^{d+1}$ nodes. Since the computations in each node of the search tree require polynomial time, we obtain an overall running time $O^\star(d^{2d})$.[2]

[2]The notation $O^\star$ suppresses polynomial factors.

### 11.2.2 Kernelization

A great tool in parameterized algorithmics is data reduction by kernelization. One can think of it as a preprocessing procedure: The problem at hand is a hard one, but it might contain some relatively easy parts. The idea is to get rid of these in a (polynomial-time) preprocessing step and to obtain the 'really hard' core, the so-called *problem kernel*, of the problem. If the size of this kernel does not depend on the input size $|\mathcal{I}|$ of the original problem any more, but is bounded by a function depending on the parameter $k$ only, we are done: applying any brute force algorithm on this hard kernel leads directly to an FPT running time where the combinatorial explosion only happens in $k$. The existence of a problem kernel whose size is bounded by a function of $k$ hence implies for a problem to be in FPT with respect to $k$, and one can show that the converse holds as well.

More formally, we say that a parameterized problem admits a problem kernel with respect to parameter $k$, if an instance $(\mathcal{I}, k)$ can be transformed in polynomial time (measured in the input size $|\mathcal{I}|$) into an equivalent instance $(\mathcal{I}', k')$ such that $|\mathcal{I}'| + k' \leqslant g(k)$ for a computable function $g$ only depending on $k$. The rules describing the transformation are then called *data reduction* rules, and the new instance $(\mathcal{I}', k')$ is called the *problem kernel*. For practical applicability, one is in particularly interested in kernels of polynomial size.

#### Example: Data Reduction and a $O(k^2)$ Kernel for Vertex Cover

For the Vertex Cover problem, one can immediately think of two easy reduction rules. Let $(G, k)$ be our input. First, it is obvious that we can safely delete isolated vertices (i.e., vertices without incident edges) from $G$, as they cannot cover any edge. Second, if there is a vertex $v \in V(G)$ having more than $k$ incident edges, then $v$ clearly has to belong to any solution $S$ of size $k$—a vertex cover *not* containing $v$ must contain all its (more than $k$) neighbors, which is too much. Hence, it is safe to put any vertex of degree strictly greater than $k$ into $S$, delete all its incident edges, and decrement the value of $k$ by one. This rule is known as the *Buss rule*. If $G$ admits a vertex cover of size $k$, then after applying these two rules exhaustively, we end up with a graph $G'$ having at most $k^2$ edges (as the Buss rule is not applicable anymore, $G'$ has maximum degree at most $k$, and hence $k$ vertices in $G'$ can cover at most $k^2$ edges) and at most $k^2 + k$ vertices (as there are no isolated vertices in $G'$). This yields a kernel of size $O(k^2)$ for Vertex Cover.

#### Example: Polynomial Kernel for Coalitional Manipulation for Copeland

Given a voting profile consisting of the voters' preference orders over the set $\mathcal{C}$ of candidates, the Coalitional Manipulation problem asks if a set of $m$ manipulators is able to make a given candidate win the election by casting their votes in an appropriate way. A *Copeland winner* of the election is a candidate who maximizes the number of candidates that he beats in pairwise comparisons (for simplicity, we assume that the number of voters is odd). Dey et al. (2016) show that if $m$ is polynomial in the number $|\mathcal{C}|$ of candidates, then Coalitional Manipulation for Copeland voting admits a polynomial kernel with respect to $|\mathcal{C}|$.

They consider the weighted majority graph of the election where vertices correspond to the candidates, and for any two candidates $x$ and $y$, the weight of the edge $(x, y)$ is the number of voters who prefer $x$ to $y$ minus the number of voters who prefer $y$ to $x$. The idea of the reduction rule is to replace large edge weights (those greater than $m$) by smaller ones ($m + 1$ or $m + 2$, so that the parity of the weight is preserved). This guarantees that each weight in the new majority graph is in $O(m)$, that the parities of the weights are unchanged, and that the Copeland score of each candidate remains the same after the application of the rule. Using a construction by McGarvey (1953), such a new majority graph can be realized by a voting profile of size $O(|\mathcal{C}|^2 \cdot m)$, giving us a kernel of size polynomial in $|\mathcal{C}|$.

We remark that Dey et al. (2016) show for the more general POSSIBLE WINNER problem that no kernel of size polynomial in $|\mathcal{C}|$ is likely to exist (cf. Section 11.4.1).

**Example: Trivial Kernel for EEF ALLOCATION**

Bliem et al. (2016) study the problem of assigning a set $\mathcal{O}$ of indivisible objects to a set $N$ of agents in a Pareto efficient and envy-free way. An instance of EEF ALLOCATION can be described as a triple $\mathcal{I} = (N, \mathcal{O}, \succsim)$ where $\succsim$ contains a preference relation $\succsim_i$ for each agent $i \in N$. The task is to find an allocation of the objects to the agents that is envy-free and Pareto efficient. Naturally, each object must be allocated to only one agent, so any allocation $\pi : N \to 2^{\mathcal{O}}$ must satisfy $\pi(i) \cap \pi(j) = \emptyset$ for any two different agents $i, j \in N$. We say that an allocation $\pi$ is *envy-free* if for any two agents $i, j$ we have $\pi(i) \succsim_i \pi(j)$. We call $\pi$ *Pareto efficient*, if there is no allocation $\pi'$ such that $\pi'(i) \succsim_i \pi(i)$ for each agent $i$, and at least one agent is strictly better off in $\pi'$ than in $\pi$ (for more precise definitions, see Bliem et al. (2016)).

Assuming *monotonic additive preferences*, each agent $i$ has a non-negative utility function $w_i \colon \mathcal{O} \to \mathbb{R}_0^+$ such that $Y \succsim_i X$ exactly if $\sum_{o \in X} w_i(o) \leqslant \sum_{o \in Y} w_i(o)$ for any two sets $X, Y$ of objects. In such a model, we can safely assume that each agent assigns a positive utility to at least one object, and similarly, each object has positive utility for at least one agent. However, this implies that if $|N| > |\mathcal{O}|$, then no allocation can be envy-free: any agent that obtains no objects at all envies at least one other agent. This yields a trivial kernel for parameter $|\mathcal{O}|$, the number of objects: if $|N| > |\mathcal{O}|$, then we can replace the instance $\mathcal{I}$ with any small 'no'-instance; otherwise, $|N| \leqslant |\mathcal{O}|$ and thus the size of the whole instance $\mathcal{I}$ is bounded by a function of the parameter $|\mathcal{O}|$.

We remark that another promising approach is to consider (the weaker concept of) *partial* kernels. Roughly speaking, this means that for problems featuring several dimensions of the input (such as the number of voters and the number of candidates in a voting problem), one can also try and reduce at least one of the dimensions such that its size only depends on the parameter value. For more details, we refer to Section 3.5 of the article by Bredereck et al. (2014).

### 11.2.3 Integer Linear Programming

Many problems can be formulated in terms of an optimization task with a linear objective function and several constraints given by linear (in)equalities. These *linear programs* can be described in their canonical form as follows:

**Linear Programming (LP):**
Input: Matrix $A \in \mathbb{R}^{m,n}$, two vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.
Task: Find a vector $x \in \mathbb{R}^n$ with $x \geqslant 0$ that fulfills $Ax \leqslant b$ and, among all such vectors, maximizes the dot product $c^T x$.

The problem can equivalently be formulated in other variants, e.g., as a minimization problem, or with equalities.

LP problems are known to be solvable in polynomial time. If the variables can only take integral values, one speaks of an *ILP (Integer Linear Program)*. This makes the problem more difficult in general: the corresponding decision problem is NP-complete (Karp, 1972). However, an ILP formulation of a problem can help us obtain an FPT result: A famous theorem by Lenstra (1983) states that solving an ILP is fixed-parameter tractable if the parameter is the number of variables or the number of constraints. Lenstra's running time was later improved by Kannan (1987) and Frank and Tardos (1987), yielding that an ILP with $p$ variables can be solved in $O(p^{2.5p+o(p)} \cdot |\mathcal{I}|)$ time, where $|\mathcal{I}|$ is the input size.

However, we shall remark that the combinatorial explosion of the running time shown by Lenstra is terrible, rendering it impractical. Lenstra's result should therefore be seen as a classification theorem in the first place. We refer to a more detailed discussion about ILP-based fixed-parameter tractability by Bredereck et al. (2014, Section 3.1).

#### Example: ILP for Vertex Cover

We start by giving a negative example for Vertex Cover. For each vertex $v \in V(G)$, we create a binary variable $x_v \in \{0,1\}$: including some vertex $v$ in the vertex cover corresponds to setting the value of variable $x_v$ to 1. The following ILP computes a minimum vertex cover for $G$:

$$\begin{array}{lll} \text{Minimize} & \sum_{v \in V(G)} x_v & \\ \text{subject to} & x_u + x_v \geqslant 1 & \forall \{u,v\} \in E(G); \\ & x_v \in \{0,1\} & \forall v \in V(G). \end{array}$$

However, the number of variables here is $|V(G)|$, and thus depends not only on the parameter $k$ but on the input size, so Lenstra's result is not applicable.

#### Example: ILP for EEF Allocation

Bliem et al. (2016) encode an instance $\mathcal{I} = (\{1,\dots,n\}, \mathcal{O}, \succsim)$ of EEF Allocation as an ILP, assuming *0/1 preferences*. In such a model, the preferences of each agent $i \in \{1,\dots,n\}$ are determined by a utility function $w_i \colon \mathcal{O} \to \{0,1\}$.

To formulate this problem as an ILP, we define the *fingerprint* of an object $o \in \mathcal{O}$ as the (binary) vector $f_o = (w_1(o), \dots, w_n(o))$; let $F = \{f_o \mid o \in \mathcal{O}\}$ be the set of

all fingerprints. We can then describe any allocation by the number $x_i^f$ of objects with fingerprint $f$ assigned to agent $i$, for any $f \in F$ and $i \in \{1, \dots, n\}$. Using the variables $x_i^f$, formulating envy-freeness is straightforward; to express efficiency, we need to observe that Pareto efficiency under 0/1 preferences is equivalent to assigning each object to an agent for whom it carries utility 1. Hence, an allocation is EEF if it fulfills the following constraints:

$$\begin{aligned}
&x_i^f = 0 && \text{for each } f \in F \text{ and } i \in \{1, \dots, n\} \text{ with } f[i] = 0;\\
&\textstyle\sum_{i=1}^{n} x_i^f = |\{o \in \mathcal{O} \mid f_o = f\}| && \text{for each } f \in F;\\
&\textstyle\sum_{f \in F} x_i^f \cdot f[i] \geqslant \sum_{f \in F} x_j^f \cdot f[i] && \text{for each } i, j \in \{1, \dots, n\} \text{ with } i \neq j.
\end{aligned}$$

As each fingerprint is a binary vector of length $n$, we get $|F| \leqslant 2^n$. The number of variables in our ILP is therefore at most $n2^n$, implying that EEF ALLOCATION is FPT with respect to parameter $n$, the number of agents.

### 11.2.4 Color-coding

Let us discuss here an elegant technique called *color-coding*, introduced by Alon et al. (1995), originally developed to solve certain cases of SUBGRAPH ISOMORPHISM. This randomized method is helpful when introducing constraints on a solution enables us to find them more easily. Instead of giving formal definitions, let us illustrate how color-coding works through the following example.

#### Example: Color-coding for MINIMAL APPROVAL VOTING

Let $\mathcal{I} = (\mathcal{E}, k, d)$ be an instance of MINIMAX APPROVAL VOTING with $\mathcal{E} = (\mathcal{C}, \mathcal{V})$ (cf. the example in Section 11.2.1). We present a randomized FPT algorithm MAV-CC with parameter $|\mathcal{V}| + k$ proposed by Misra et al. (2015); see Algorithm 3. Let us assume that $\mathcal{I}$ is a 'yes'-instance, and $S$ is a solution committee of size $k$.

**Algorithm 3** Color-coding algorithm for MINIMAX APPROVAL VOTING

**procedure** MAV-CC($\mathcal{E} = (\mathcal{C}, \mathcal{V}), d, k$)
  choose a coloring $\kappa : \mathcal{C} \to \{1, \dots, k\}$ randomly;
  **for all** $v \in \mathcal{V}$ **do**
    choose a set $X_v \subseteq \{\kappa(x) \mid x \in v\}$ with $|X_v| \geqslant (|k + |v| - d)/2$ randomly;
  $S' \leftarrow \emptyset$;
  **for all** $c \in \{1, \dots, k\}$ **do**
    $A_c \leftarrow \bigcap\{\text{candidates of } v \text{ with color } c \mid v \in \mathcal{V}, c \in X_v\}$;
    **if** $A_c \neq \emptyset$ **then** put any $a_c \in A_c$ into $S'$;
  **if** $|S'| = k$ **then** output $S'$;
  **else** output 'no';

First, we color our candidates with $k$ colors randomly (independently, with a uniform distribution). Given a coloring $\kappa : \mathcal{C} \to \{1, \dots, k\}$, for each color $c \in \{1, \dots, k\}$ we define the *color class* $\mathcal{C}_c = \{x \in \mathcal{C} \mid \kappa(x) = c\}$ as the set of candidates receiving color $c$. We call the coloring $\kappa$ *good*, if it makes our solution $S$ *colorful*,

meaning that $S \cap \mathcal{C}_c \neq \emptyset$ for each color $c$; clearly, a colorful solution must contain *exactly* one candidate from each color class.

Let us assume that $\kappa$ is a good coloring. Next, for each vote $v \in \mathcal{V}$ we guess the set $X_v$ of *consensus colors* for $v$, containing the colors of the candidates in $v \cap S$. Notice that the consensus colors for a vote determines its distance from $S$, namely $\text{dist}(S, v) = |S \setminus v| + |v \setminus S| = k + |v| - 2|X_v|$. Hence, we can immediately discard those guesses where $\text{dist}(S, v) > d$ for some vote $v \in \mathcal{V}$.

Given the consensus colors for each vote, finding a colorful solution is easy: for each color $c$, we need to check whether all sets $v \cap \mathcal{C}_c$ where $v \in \mathcal{V}, c \in X_v$ have at least one common candidate. If so, we put one such candidate $a_c$ into our solution $S'$ for each $c$. Observe that $S'$ is indeed a solution, as each vote contains at least $|X_v|$ candidates from $S'$. Since $S$ always yields some $a_c \in S$ contained in all votes which have $c$ as a consensus color, we are bound to find a solution (supposing we guessed the consensus colors correctly).

Let us consider the running time of MAV-CC. Given a good coloring, we need to consider all possible consensus color sets for each vote, which means $(2^k)^n$ possibilities (where $n = |\mathcal{V}|$). For each of these cases, looking for a solution takes $O(kn|\mathcal{C}|)$ time. But how can we obtain a good coloring? Clearly, our random coloring $\kappa$ is good with probability $\frac{k!}{k^k} \geqslant e^{-k}$. Thus, repeating the whole procedure $e^k$ times guarantees that we will obtain a good coloring, and hence a solution, with high probability. This yields a total running time of $e^k 2^{nk} O(kn|\mathcal{C}|)$, which is fixed-parameter tractable with respect to parameter $n + k$.

To de-randomize the above algorithm, we need to deterministically construct a family of coloring functions such that any given committee of size $k$ becomes colorful in at least one of the colorings. This can be achieved by so-called $k$-*perfect families of hash functions*; an explicit construction of such a family of size $e^k k^{O(\log k)} \log |\mathcal{C}|$ is given by Naor et al. (1995).

### 11.2.5 Multiple Parameters

For a truly detailed insight into a problem's computational complexity, one can typically determine several parameters that might influence its tractability. Allowing for multiple parameters is thus crucial in the parameterized framework.

Suppose we want to handle $t$ parameters in our problem, so each instance $\mathcal{I}$ is associated with parameters $k_1, \ldots, k_t$. Intuitively, a fixed-parameter tractable algorithm in such a model is one that runs in time $f(k_1, \ldots, k_t)|\mathcal{I}|^{O(1)}$ for some computable function $f$. However, instead of extending the formalism of the original framework, it suffices to define the parameter as the $t$-tuple $(k_1, \ldots, k_t)$; such composite parameters are usually called *combined parameters*. Equivalently, we can simply define the sum $k_1 + \cdots + k_t$ or the maximum $\max\{k_1, \ldots, k_t\}$ as the parameter; either of these choices yields the same notion of fixed-parameter tractability.

To exploit the full power of parameterized complexity, allowing for multiple parameters is just the first step. Looking for an FPT algorithm with parameters $k_1, \ldots, k_t$ amounts to searching for an algorithm that is efficient if *all* of the values $k_1, \ldots, k_t$ are small. A much more informative approach is to adopt a *multidimensional* view (also called *multivariate algorithmics*; see the paper by Niedermeier

(2010)), and regard each value $k_i$ as either (i) a fixed constant, (ii) a parameter, or (iii) unbounded. This yields $3^t$ variants of the original problem, and determining the (parameterized) complexity for each of these variants offers a detailed landscape of its computational tractability.

A nice example for such a multidimensional analysis is the work by De Haan (2016a) who investigated the complexity of judgment aggregation based on the Kemeny rule with respect to five parameters and all their possible combinations.

## 11.3 Parameterized Intractability

In the previous sections, we have gotten to know some basic techniques for showing fixed-parameter tractability of a hard problem. But what can we do if none of these techniques seems to be applicable to our problem at hand? There are many more techniques that we could give a try; see the literature referred to in the introduction. However, it might also happen that the problem on hand simply is *not* in FPT. For such cases, we can try to provide evidence that the problem does not admit an FPT algorithm: similarly to the theory of NP-hardness, parameterized complexity offers an intractability theory which provides the possibility to compare the computational hardness of parameterized problems. For a more detailed view on this topic, we refer to the books by Downey and Fellows (1999, 2013), and by Flum and Grohe (2006).

Before we start, let us introduce two well-known notions from graph theory. Given a graph $G$, an *independent set* is a set $I \subseteq V(G)$ of vertices in $G$ such that no two of them are adjacent to each other in $G$. A *clique* is a set $C \subseteq V(G)$ of vertices that are pairwise adjacent in $G$. The notions of vertex cover, independent set, and clique are closely related: $S \subseteq V(G)$ is a vertex cover of $G$ if and only if $I := V(G) \setminus S$ is an independent set of $G$, which in turn holds if and only if $I$ is a clique in the complement graph $\overline{G}$.[3]

| INDEPENDENT SET: | |
|---|---|
| Input: | An undirected graph $G$, and an integer $k$. |
| Question: | Does $G$ contain an independent set of size at least $k$? |

| CLIQUE: | |
|---|---|
| Input: | An undirected graph $G$, and an integer $k$. |
| Question: | Does $G$ contain a clique of size at least $k$? |

The two problems above were proved to be NP-complete by Karp (1972), so their classical complexity is the same as that of VERTEX COVER. However, when parameterized by $k$, they exhibit a great difference: despite the relentless effort to design efficient algorithms for these problems, neither CLIQUE nor INDEPENDENT SET has been shown to admit an FPT algorithm.

[3]The complement graph of $G$ is the graph $\overline{G}$ which has the same set of vertices as $G$, and there is an edge between two different vertices in $\overline{G}$ if and only if there is no edge between them in $G$.

### 11.3.1 FPT Reduction

In classical complexity theory, a polynomial-time many-to-one (or Karp) reduction from problem $P$ to problem $P'$ transforms—in polynomial time—an instance $\mathcal{I}$ of $P$ into an equivalent instance $\mathcal{I}'$ of $P'$, meaning that $\mathcal{I}$ is a 'yes'-instance of $P$ if and only if $\mathcal{I}'$ is a 'yes'-instance of $P'$. We now describe a similar notion of reduction for parameterized problems that can transfer fixed-parameter tractability.

Let $P$ and $P'$ be two parameterized problems. An *FPT reduction* (also called *parameterized reduction*) from $P$ to $P'$ is an algorithm that runs in FPT time (i.e., in time $f(k) \cdot |\mathcal{I}|^{O(1)}$ for a computable function $f$) and transforms an instance $(\mathcal{I}, k)$ of $P$ into an equivalent instance $(\mathcal{I}', k')$ of $P'$ such that $k' \leqslant g(k)$ for some computable function $g$. The difference from a polynomial-time reduction is thus two-fold: we have to ensure that the parameter of the new instance only depends on the original parameter, but the transformation may take FPT time. The key property of an FPT reduction is the following: if $P' \in \mathsf{FPT}$, then an FPT reduction from $P$ to $P'$ implies $P \in \mathsf{FPT}$ as well.

The classical polynomial-time reduction from Independent Set to Clique transforms an instance $(G, k)$ of Independent Set into an equivalent instance $(\overline{G}, k)$ of Clique. Thus, if we regard $k$ as the parameter in both problems, then this transformation becomes an FPT reduction. Applying the same reduction the other way around shows that Clique to Independent Set are equally hard, even in the parameterized form.

By contrast, the classical polynomial-time reduction from Independent Set to Vertex Cover is not an FPT reduction: it transforms an instance $(G, k)$ of Independent Set into an equivalent instance $(G, k' = |V(G)| - k)$ of Vertex Cover, but the new parameter $k'$ depends not only on $k$ but also on $|V(G)|$. Hence, this does *not* prove fixed-parameter tractability of Independent Set.

### 11.3.2 Parameterized Complexity Classes

Parameterized complexity offers a whole hierarchy of hardness classes, called the *weft hierarchy*, based on weighted variants of the satisfiability problem for Boolean circuits. It contains the classes $\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \cdots \subseteq \mathsf{W[t]} \subseteq \cdots \subseteq \mathsf{W[SAT]} \subseteq \mathsf{W[P]}$, where all inclusions are believed to be strict. All these classes are closed under FPT reductions, and are contained in the class XP of *slicewise polynomial* problems, containing parameterized problems that, given an instance $(\mathcal{I}, k)$, can be solved in time $|\mathcal{I}|^{f(k)}$, where $f$ is a computable function only depending on $k$. The class XP is known to be strictly larger than FPT.

| Weighted 2-CNF-Satisfiability (W-2-CNF-SAT): | |
|---|---|
| Input: | A Boolean formula $F$ in conjunctive normal form (CNF) with at most two literals per clause, and an integer $k$. |
| Question: | Does $F$ have a satisfying truth assignment of weight exactly $k$, i.e., with exactly $k$ variables set to true? |

The class W[1] contains all problems that can be reduced to the above defined W-2-CNF-SAT problem (parameterized by $k$) by an FPT reduction. If *all* problems

in W[1] are FPT reducible to a parameterized problem $P$, then $P$ is called W[1]-*hard*. If in addition $P$ is contained in W[1], then it is W[1]-*complete*.

To prove that $P$ is W[1]-hard, and consequently is not likely to admit an FPT algorithm, it suffices to provide an FPT reduction from an already known W[1]-hard problem to $P$. Besides W-2-CNF-SAT—which is W[1]-complete by definition—both CLIQUE and INDEPENDENT SET are W[1]-complete with respect to the parameter $k$ (Downey and Fellows, 1999). The following useful variant of CLIQUE is also W[1]-complete with respect to the number $k$ of colors.

| MULTICOLORED CLIQUE: | |
|---|---|
| Input: | An undirected graph $G$ whose vertices are colored with $k$ colors. |
| Question: | Is there a clique in $G$ containing one vertex from each color class? |

As another example, UNARY BIN PACKING, defined as below, is also W[1]-hard with respect to the number $b$ of bins, even if the total weight of the items equals the total bin capacity, i.e., $\sum_{i=1}^{m} w_i = b \cdot C$ (Jansen et al., 2013).

| UNARY BIN PACKING: | |
|---|---|
| Input: | Positive integers $w_1, \ldots, w_m$, $b$, $C$, all in unary encoding. |
| Question: | Is there a packing of $m$ items with weights $w_1, \ldots, w_m$ to $b$ bins such that each bin has total weight at most $C$? |

Analogously to W[1], the definition of the class W[2] is based on the WEIGHTED CNF-SATISFIABILITY problem where the clauses of the input formula can be of any size. A typical W[2]-complete problem is DOMINATING SET, asking if a graph $G$ has a *dominating set* of size $k$, i.e., a subset $D \subseteq V(G)$ of $k$ vertices such that each vertex in $V(G) \setminus D$ is adjacent to at least one vertex of $D$; the parameter is $k$.

| DOMINATING SET: | |
|---|---|
| Input: | An undirected graph $G$, and an integer $k$. |
| Question: | Does $G$ contain a dominating set of size at most $k$? |

**Example: W[1]-hardness of EEF ALLOCATION**

Bliem et al. (2016) give a parameterized reduction from UNARY BIN PACKING to show W[1]-hardness for EEF ALLOCATION with respect to the number of agents (see the examples in Sections 11.2.2 and 11.2.3 for the definitions), even if agents express their utilities encoded in unary. Given an instance $(w_1, \ldots, w_m, b, C)$ of UNARY BIN PACKING with $\sum_{i=1}^{m} w_i = b \cdot C$, we construct an instance of EEF ALLOCATION as follows: the set of objects is $\{o_1, \ldots, o_m\}$, and there are $b$ agents with identical preferences, each assigning utility $w_i$ to object $o_i$, for $i \in \{1, \ldots, m\}$.

Observe that an allocation is Pareto efficient and envy-free exactly if the total utility assigned to each agent is $(\sum_{i=1}^{m} w_i)/b = C$: each agents needs to be assigned the same total utility, and each object must be allocated. Therefore, an EEF-allocation of the objects to the $b$ agents immediately gives a packing of all items

into the $b$ bins respecting the bin capacities (where assigning object $o_i$ to the $j$-th agent corresponds to packing the item $w_i$ into the $j$-th bin), and vice versa.

Clearly, the instance of EEF ALLOCATION can be constructed in polynomial time, and the parameter $b$ in the UNARY BIN PACKING instance equals the number of agents in the constructed EEF ALLOCATION instance. Thus, we obtain W[1]-hardness with respect to the number of agents. We remark that if agents express their utilities in binary encoding, then the problem is NP-hard already for two agents (Bouveret and Lang, 2008).

## 11.4 Advanced Techniques

Here we briefly mention a few of the more advanced techniques of parameterized complexity which can help us investigate the complexity of a computationally hard problem. For further reading, we refer to the book by Cygan et al. (2015) on lower bounds for kernelization and on lower bounds assuming ETH; the latter topic is also covered by Lokshtanov et al. (2011). The survey by Marx (2008) offers an excellent summary on the connection between fixed-parameter tractability and approximation. For an extension of the parameterized framework dealing with problems beyond NP, see the recent PhD thesis by De Haan (2016b).

### 11.4.1 Lower Bounds for Kernelization

As already mentioned in Section 11.2.2, a parameterized problem is FPT if and only if it admits a kernel. But an exponential-size kernel may not be very useful in practice, and so the more interesting question is whether a given (FPT) problem admits a kernel of polynomial size.

Recently the field of kernelization has undergone exciting improvements: a series of new results have established a framework for proving lower bounds for the existence of polynomial kernels. This breakthrough started with a paper by Fortnow and Santhanam (2008), followed by Bodlaender et al. (2009) who proved the following: if a parameterized problem $Q$ whose unparameterized version is NP-hard admits an OR-composition, then it does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly (considered very unlikely in complexity theory). Here, an *OR-composition* for $Q$ is an algorithm that, given $t$ instances $(\mathcal{I}_1, k), \ldots, (\mathcal{I}_t, k)$ of $Q$, in time polynomial in $\sum_{j=1}^{t} |\mathcal{I}_j| + k$ computes a new instance $(\mathcal{I}', k')$ with $k' = k^{O(1)}$ such that $(\mathcal{I}', k') \in Q$ if and only if $(\mathcal{I}_j, k) \in Q$ for at least one index $j \in \{1, \ldots, t\}$.

*OR-cross-compositions* offer a more flexible method (Bodlaender et al., 2011a). Roughly speaking, an OR-cross-composition algorithm takes as input $t$ instances of *any* NP-hard problem $L$, and produces an instance $(\mathcal{I}, k)$ such that $(\mathcal{I}, k) \in Q$ if and only if one of the $t$ instances is in $L$; the parameter $k$ must be polynomially bounded in the maximum size of the input instances plus $\log t$. Using this extended framework, Bliem et al. (2016) proved that EEF-ALLOCATION with monotonic dichotomous preferences parameterized by the number of objects does not admit a polynomial kernel (unless NP $\subseteq$ coNP/poly).

We remark that instead of OR-(cross-)compositions, one can also use AND-(cross-)compositions, defined analogously, due to a result by Drucker (2012).

Bodlaender et al. (2011b) proposed another tool that can be used to prove the non-existence of polynomial kernels. Given two parameterized problems $Q$ and $Q'$, a *polynomial parameter transformation* (PPT) from $Q$ to $Q'$ is a function that for any instance $(\mathcal{I}, k)$ of $Q$ computes in polynomial time an equivalent instance $(\mathcal{I}', k')$ of $Q'$, where $k'$ is bounded by a fixed polynomial of $k$. Essentially, if there is a PPT from $Q$ to $Q'$ and $Q$ admits no polynomial kernel, then $Q'$ does not admit a polynomial kernel either.[4] Using this concept, Dey et al. (2016) showed for various voting rules that POSSIBLE WINNER is not likely to admit a polynomial kernel if the parameter is the number of candidates.

Finally, let us mention the technique of *weak compositions* by Dell and van Melkebeek (2010); Hermelin and Wu (2012); Dell and Marx (2012) which can be used to derive more refined lower bounds, ruling out not polynomial kernels in general, but kernels of a certain size (such as, say, a linear kernel).

### 11.4.2 Lower Bounds Assuming ETH

Impagliazzo et al. (2001) formulated the Exponential Time Hypothesis (ETH) that, roughly speaking, states that 3-SAT cannot be solved in subexponential time. Assuming ETH, one can obtain stronger lower bounds for various computational problems than only assuming the weaker assumption $\mathsf{P} \neq \mathsf{NP}$.

As shown by Chen et al. (2006), ETH implies that CLIQUE, INDEPENDENT SET, and DOMINATING SET cannot be solved in $f(k)n^{o(k)}$ time for any function $f$ on $n$-vertex graphs with parameter $k$. This shows that ETH is a stronger assumption than $\mathsf{W}[1] \neq \mathsf{FPT}$. One can obtain lower bounds also for problems in FPT; as an example, Cai and Juedes (2003) proved that VERTEX COVER cannot be solved in $2^{o(k)}n^{O(1)}$ time on an $n$-vertex graph with parameter $k$, unless ETH fails.

Such lower bounds can be transferred by appropriate reductions; the obtained lower bound depends on how the reduction changes the parameter. With this method, Cygan et al. (2016) proved that MINIMAX APPROVAL VOTING admits no algorithm running in $O^\star(2^{o(d \log d)})$ time, showing that the algorithm by Misra et al. (2015) described in Section 11.2.1 is essentially optimal.

### 11.4.3 Approximation and Parameterized Algorithms

Approximation algorithms have a polynomial running time, but produce only suboptimal solutions; by contrast, parameterized algorithms provide optimal solutions, but at a cost of increased running time. Combining these two approaches yields a variety of methods to deal with computationally hard problems; here we only highlight a few ideas and results connected to computational social choice.

Given an optimization problem $Q$, we can ask whether $Q$ admits an *FPT-approximation algorithm*: one that produces an approximate solution and runs in FPT time with respect to a given parameter. This idea can be extended to *FPT-approximation schemes*; an example is the algorithm by Cygan et al. (2016) for MINIMAX APPROVAL VOTING that for any $\varepsilon > 0$ in time $O^\star((3/\varepsilon)^{2d})$ produces an $\varepsilon$-approximation (i.e., a committee with distance at most $(1+\varepsilon)d$ from any vote).

[4] In fact, we also need a polynomial reduction from $Q'$ to $Q$, which is guaranteed if $Q$ is NP-hard and $Q'$ is in NP.

Another strong connection between approximation schemes and parameterized complexity was observed by Bazgan (1995), and independently, Cesati and Trevisan (1997): if $Q$ admits an EPTAS[5], then deciding whether an instance of $Q$ admits a solution with value at least (or, if $Q$ is a minimization problem, at most) $k$ is FPT with parameter $k$. This fact is often used in negative form: e.g., as observed by Gurski and Roos (2014), the W[2]-hardness of constructive control by adding/deleting candidates in Lull or Copeland elections implies that these problems do not admit an EPTAS, unless $\mathsf{W}[2] = \mathsf{FPT}$.

### 11.4.4 Parameterized Complexity for Problems Beyond NP

Recently, a new theoretical framework has been developed that combines the idea of parameterized complexity with the practice of converting hard problems into well-studied, standardized problems like SAT and using already existing solvers to deal with the transformed instance; the thesis by De Haan (2016b) offers a thorough introduction to this framework. SAT solvers are highly efficient in practice, but their use is limited by the fact that only problems belonging to NP admit a polynomial-time reduction to SAT. Hence, for problems that lie in higher classes of the Polynomial Hierarchy (so, are 'beyond' NP), it might be helpful to allow transformations into SAT that take FPT time with respect to some parameter.

This idea can be formalized in different ways, leading to various new methods and complexity classes; we only mention two prominent concepts. The first is to consider parameterized problems that admit a many-to-one FPT reduction to SAT; the corresponding complexity class is called para-NP, a direct parameterized analog of NP.[6] Another possibility is to use Turing reductions instead of many-to-one reductions when converting a parameterized problem into SAT. This leads to the definition of complexity classes like $\mathsf{FPT}^{\mathsf{NP}}$[few], containing problems solvable by an FPT algorithm that has access to a SAT oracle, with the restriction that the number of oracle queries must be upper-bounded by a function of the parameter. An example from social choice is the agenda safety problem for the majority rule in judgment aggregation which was shown to be $\mathsf{FPT}^{\mathsf{NP}}$[few]-complete with respect to the agenda size as parameter by Endriss et al. (2015).

## 11.5 Conclusion

This chapter is meant to be a gentle introduction to parameterized complexity for researchers in computational social choice. We have presented the basic approach, several standard techniques and ideas, and tried to give some simple examples that allow for a quick start in parameterized complexity. We also wanted to give some glimpse of what there is beyond the basic techniques in order to provide a good starting point for the interested reader to get into this

[5] An *Efficient Polynomial-Time Approximation Scheme* or EPTAS is an algorithm that produces an $\varepsilon$-approximate solution in $f(\varepsilon) \cdot |\mathcal{I}|^{O(1)}$ time for any instance $\mathcal{I}$.

[6] The class para-NP can be alternatively defined as the set of parameterized problems solvable by a nondeterministic Turing machine in FPT time.

beautiful research area. In this context, we also refer to the collection of PhD theses in computational social choice (www.illc.uva.nl/COMSOC/theses.html)—several of them use parameterized complexity and can serve as a signpost for future directions and trends.

The area of parameterized complexity has experienced an immense boom in recent years, and many new results and techniques have made this area a very active and exciting one. We are convinced that this development also means a big chance for the analysis of problems from computational social choice. Shedding some more light on the complexity landscape of hard problems, parameterized complexity does not only offer a possibility to face the criticism that complexity theory is only a worst-case analysis and hence not suitable to serve as a barrier against manipulative behavior. It might also contribute to make you see the many faces of a hard problem and to gain a better feeling of what really makes it hard. We admit that parameterized complexity cannot always save you from having a hard time with your problems—but probably it can make you enjoy them more!

## Acknowledgment

We thank Ulle Endriss for inviting us to write this chapter, and we are indebted to Ronald de Haan for carefully reading a preliminary version of it and providing us with helpful advice. Ildikó Schlotter was supported by the Hungarian Scientific Research Fund (OTKA grants no. K-108383 and no. K-108947).

# Bibliography

N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

C. Bazgan. Schémas d'approximation et complexité paramétrée. Technical report, Université Paris Sud, 1995. Mémoire de DEA.

N. Betzler, R. Bredereck, J. Chen, and R. Niedermeier. Studies in computational aspects of voting—a parameterized complexity perspective. In H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, pages 318–363, Berlin, Heidelberg, 2012. Springer.

B. Bliem, R. Bredereck, and R. Niedermeier. Complexity of efficient and envy-free resource allocation: Few agents, resources, or utility levels. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 102–108, 2016.

H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8): 423–434, 2009.

H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 165–176, 2011a.

H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011b.

S. Bouveret and J. Lang. Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. *Journal of Artificial Intelligence Research*, 32(1):525–564, 2008.

R. Bredereck, J. Chen, P. Faliszewski, J. Guo, R. Niedermeier, and G. J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Science and Technology*, 19(4):358–373, 2014.

L. Cai and D. W. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.

M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997.

J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8): 1346–1367, 2006.

J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.

M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

M. Cygan, Ł. Kowalik, A. Socała, and K. Sornat. Approximation and parameterized complexity of minimax approval voting. *CoRR*, abs/1607.07906, 2016. arXiv:607.07906 [cs.DS].

H. Dell and D. Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 68–81, 2012.

H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 251–260, 2010.

P. Dey, N. Misra, and Y. Narahari. Kernelization complexity of possible winner and coalitional manipulation problems in voting. *Theoretical Computer Science*, 616:111–125, 2016.

R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, Heidelberg, 2005.

R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, 1999.

R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, London, 2013.

A. Drucker. New limits to classical and quantum instance compression. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 609–618, 2012.

U. Endriss, R. de Haan, and S. Szeider. Parameterized complexity results for agenda safety in judgment aggregation. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 127–136, 2015.

P. Faliszewski and R. Niedermeier. Parameterization in computational social choice. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*. Springer, 2015.

J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, New York, 2006.

L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 133–142. ACM, 2008.

A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

F. Gurski and M. Roos. Binary linear programming solutions and non-approximability for control problems in voting systems. *Discrete Applied Mathematics*, 162:391–398, 2014.

R. de Haan. Parameterized complexity results for the Kemeny rule in judgment aggregation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1502–1510, 2016a.

R. de Haan. *Parameterized Complexity in the Polynomial Hierarchy*. PhD thesis, Technische Universität Wien, 2016b.

D. Hermelin and X. Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 104–113, 2012.

R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.

R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

C. Lindner and J. Rothe. Fixed-parameter tractability and parameterized complexity applied to problems from computational social choice. In A. Holder, editor, *Mathematical Programming Glossary*. INFORMS Computing Society, 2008.

D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of European Association for Theoretical Computer Science*, 105:41–71, 2011.

D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

D. C. McGarvey. A theorem on the construction of voting paradoxes. *Econometrica*, 21:608–610, 1953.

N. Misra, A. Nabeel, and H. Singh. On the parameterized complexity of minimax approval voting. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 97–105, 2015.

M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191, 1995.

R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 17–32, 2010.