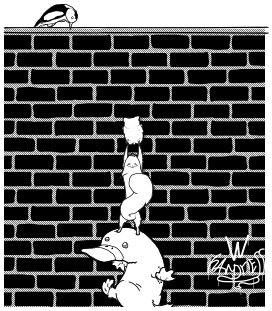
Chapter 13
Fixed-parameter intractability

In this chapter, we use parameterized reductions and W[1]*-hardness to give evidence that certain parameterized problems are unlikely to be fixed-parameter tractable.*



The goal of the previous chapters was to present the most fundamental algorithmic techniques for showing the fixed-parameter tractability of parameterized problems. In this chapter, we study the complementary questions on lower bounds: how it is possible to show (give evidence) that a parameterized problem is not fixed-parameter tractable.

In the earliest days of computer science, when the first algorithms were developed to solve combinatorial problems of practical interest such as the Traveling Salesperson Problem, it was observed with frustration that every proposed algorithm had an inconveniently large worst-case upper bound on the running time, typically exponential in the size of the input. Several decades of research have not been able to explain and prove why exponential time is needed for these problems. The theory of NP-completeness at least showed that there is one common underlying reason for the lack of polynomial-time algorithms and therefore conditional lower bounds based on NP-completeness are the best we can have at this point of time. The fact that computation cannot be cheap seems to be a fundamental principle and understanding the exact reason for this apparent hardness is a deep question that is well beyond our current knowledge.

Our goal in this chapter is to develop a lower bound theory for parameterized problems, similar to the NP-completeness theory of polynomial-time computation. Instead of taking this as an opportunity to delve deeper into the mysteries of efficient computation, we adopt the pragmatic viewpoint of the algorithm designer: our focus is on presenting evidence for as many problems

as possible that algorithms with certain specifications do not exist. Being aware of and being able to produce such negative results is essential for the algorithm designer: without them, countless hours can be wasted on trying to prove results that contradict commonly accepted assumptions. Knowing that certain problems are (probably) not fixed-parameter tractable prevents us from attacking the same problem over and over again with little hope of progress. Furthermore, the lower-bound theory helps the algorithm designer in a more subtle way as well. In many cases, it is very helpful to look at a problem from both the algorithmic and the complexity viewpoint at the same time. A failed attempt at finding an algorithm can highlight certain difficult situations, giving insight into the structure of hard instances, which can be the starting point of a hardness proof. Conversely, if one can pinpoint a specific reason why a proposed hardness proof cannot be made to work, then it may suggest an algorithmic idea that can renew the attack on the algorithmic side of the question. The authors of this textbook have experienced many times that an answer was found only after repeatedly jumping back and forth between attacks from the algorithmic side and the complexity side of the problem. Moreover, the hardness proofs might tell us what special cases or problem variants merit further exploration. Hence the lower bound theory helps not only in finding the answers, but can even steer the algorithm designer towards the right formulation of the questions.

As we have no proof of $\mathrm{P} \neq \mathrm{NP}$, we cannot rule out the possibility that problems such as Clique and Dominating Set are polynomial-time solvable and hence FPT. Therefore, our lower bound theory has to be conditional: we are proving statements of the form "if problem A has a certain type of algorithm, then problem B has a certain type of algorithm as well." If we have accepted as a working hypothesis that B has no such algorithms (or we have already proved that such an algorithm for B would contradict our working hypothesis), then this gives evidence that problem A does not have this kind of algorithms either. To prove such statements in the context of fixed-parameter tractability, we need a notion of reduction that transfers (negative evidence for) fixed-parameter tractability from one problem to the other. As we shall see, the standard notion of polynomial-time reduction used in NP-completeness theory is not sufficient for our purposes. Section 13.1 introduces the notion of parameterized reductions, which have a slightly different flavor than NP-hardness proofs and therefore require a different toolbox of techniques. Then Section 13.2 presents a selection of basic reductions from Clique to various problems. If we accept as a working hypothesis that Clique is not fixed-parameter tractable, then these reductions from Clique are practical evidence that certain other problems are not fixed-parameter tractable either. The assumption that Clique is not fixed-parameter tractable is a stronger assumption than $\mathrm{P} \neq \mathrm{NP}$, but it seems that we need such a strong assumption, as we currently do not know how to base fixed-parameter intractability results on assuming $\mathrm{P} \neq \mathrm{NP}$ only.

A remarkable aspect of NP-completeness is that there are literally thousands of natural hard problems that are equally hard in the sense that they are reducible to each other. The situation is different in the case of parameterized problems: there seem to be different levels of hardness and even basic problems such as Clique and Dominating Set seem to occupy different levels. Downey and Fellows introduced the W-hierarchy in an attempt to classify parameterized problems according to their hardness. We give a brief overview of this hierarchy in Section 13.3. The Clique problem is W[1]-complete, that is, complete for the first level of the W-hierarchy. Therefore, Clique not being fixed-parameter tractable is equivalent to FPT $\neq$ W[1]. This is the basic assumption of parameterized complexity; we interpret W[1]-hardness as evidence that a problem is not fixed-parameter tractable. In Section *13.4, we connect the class W[1] to a fundamental computational problem about Turing machines. This connection gives further evidence supporting the assumption FPT $\neq$ W[1]. Moreover, it allows a very convenient way of proving membership in W[1]. In Section *13.5, we use this and other arguments to conclude that all the problems studied in Section 13.2 are actually complete either for W[1] or for W[2]. In Section 13.6, we finish the chapter with a selection of W[1]-hardness results, demonstrating some of the challenges one faces in typical hardness proofs and ways of overcoming these difficulties. We remark that Section 14.4.1 of the next chapter contains further important parameterized reductions.

Following our pragmatic viewpoint, we omit here the discussion of issues that belong to structural complexity theory and hence are not essential to the algorithm designer. What complexity classes can be defined and which problems are complete for them? Which of these classes are equal? Are there intermediate problems between two classes? What notions of reductions can we define? Can we increase the strength of our negative evidence by basing it on weaker conjectures? What role non-uniformity plays in the complexity of the problems? These are fine theoretical questions, but are not of immediate importance to the algorithm designer who only wants some evidence that the answer to a particular algorithmic question is negative.

Finally, let us comment on the fact that all the lower bounds of this and the following chapters are conditional on various conjectures. The reader might be skeptical of the validity of these lower bounds, as they are based on unproven assumptions. In particular, the strongest of these conjectures, SETH, is still somewhat controversial and not fully accepted by the computational complexity community. Nevertheless, these lower bounds still send an important and useful message. The reader might dismiss these lower bounds and spend time on working towards an algorithmic result violating these lower bounds. But then the reader should be aware that he or she is actually working towards disproving one of the basic conjectures and the difficulty of proving the algorithmic result is not specific to the problem at hand, but it requires answering a basic well-studied question first. Then one could argue that perhaps it is more efficient to spend time on concentrating on this basic question

directly, rather than answering it indirectly via some more problem-specific algorithmic question.

13.1 Parameterized reductions

Let us recall the standard notion of polynomial-time reduction used in NP-hardness proofs. A *polynomial-time many-one*[1] *reduction* from problem A to problem B is a polynomial-time algorithm that, given an instance x of problem A, outputs an *equivalent* instance x' of problem B, that is, x is a yes-instance of problem A if and only if x' is a yes-instance of problem B. If there is such a reduction from A to B and B is polynomial-time solvable, then A is also polynomial-time solvable: given an instance x of problem A, we can run the reduction to get an instance x' of problem B, which we can solve using the assumed polynomial-time algorithm for B.

We need an analogous notion of reduction for parameterized problems that transfers fixed-parameter tractability.

Definition 13.1 (parameterized reduction). Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a yes-instance of A if and only if (x', k') is a yes-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f.

As a minor technical remark, let us observe that it can be assumed that the functions f and g are nondecreasing: for example, we may replace the computable function $g(k)$ with $\hat{g}(k) = \max_{i=1}^{k} g(k) \geq g(k)$, which is a computable nondecreasing function.

Before comparing this notion of reduction to classic polynomial-time reductions, let us formally state and prove that parameterized reductions work as intended.

Theorem 13.2. *If there is a parameterized reduction from A to B and B is* FPT*, then A is* FPT *as well.*

Proof. Let (x, k) be an instance of problem A. The parameterized reduction gives an equivalent instance (x', k') in time $f(k)|x|^{c_1}$ with $k' \leq g(k)$ and $|x'| \leq f(k)|x|^{c_1}$ (clearly, the running time of the reduction is an upper bound on the size of the instance created). Suppose that B has an algorithm with

[1] The terminology "many-one" refers to the fact that many instances of problem A might be mapped to the same instance of problem B. Some authors use the term *Karp-reduction* for such a reduction.

running time $h(k)n^{c_2}$; by the earlier discussion, we may assume that h is nondecreasing. Then running this algorithm on (x', k') decides whether (x', k') is a yes-instance of B (equivalently, whether (x, k) is a yes-instance of A) in time at most

$$h(k')|x'|^{c_2} \leq h(g(k))(f(k)|x|^{c_1})^{c_2}$$

(the inequality uses that h is nondecreasing). Together with the time required for the parameterized reduction, the total running time is at most $f'(k)|x|^{c_1 c_2}$, where $f'(k) = f(k) + h(g(k))f(k)^{c_2}$ is a computable function. That is, the problem B is fixed-parameter tractable. □

The second item in Definition 13.1 requires something that is not required for classic polynomial-time reductions: the new parameter should be bounded by a function of the parameter of the original instance. Therefore, not every NP-hardness proof gives a parameterized reduction.

There is a very simple reduction from Independent Set to Clique: the size of the maximum independent set in graph G is the same as the size of the maximum clique in the complement $\overline{G}$. Therefore, given an instance (G, k), creating an instance $(\overline{G}, k)$ is a polynomial-time reduction from Independent Set to Clique. In fact, this is a parameterized reduction: the new parameter is the same in the original instance, hence $g(k) = k$ is a trivial bound on the new parameter. The same reduction can be used in the other direction as well, that is, to reduce Clique to Independent Set. Therefore, these two problems are equally hard in the sense that one is FPT if and only if the other is.

It is a well-known fact that an n-vertex graph G has an independent set of size k if and only if it has a vertex cover of size $n - k$. Therefore, (G, k) is a yes-instance of Independent Set if and only if $(G, n-k)$ is a yes-instance of Vertex Cover. This immediately gives a polynomial-time reduction from Independent Set to Vertex Cover. However, this is *not* a parameterized reduction: as n can be arbitrarily large compared to k, there is no function $g(k)$ such that $n - k \leq g(k)$. We do not expect that there is a parameterized reduction from Independent Set to Vertex Cover: as the latter problem is FPT, the existence of such a reduction would imply by Theorem 13.2 that Independent Set is FPT as well, which we do not believe.

The third item in Definition 13.1 allows that the running time of the reduction is more than polynomial: it can have an additional factor depending only on the parameter k. This means that not every parameterized reduction is a polynomial-time reduction and if there is a parameterized reduction from A to B such that the unparameterized version of A is NP-hard, then this *does not* necessarily imply that B is NP-hard. As an artificial example, consider the problem $\text{Clique}_{\log}$, where (G, k) is a yes-instance if $k \leq \log_2 |V(G)|$ and G has a clique of size k. The following simple transformation is a parameterized

reduction from Clique to $\text{Clique}_{\log}$: given an instance (G, k), we output the instance (G', k), where G' is obtained from G by adding 2^k isolated vertices. Then $k \leq \log_2 |V(G')|$ always holds, and hence (G', k) is a yes-instance of $\text{Clique}_{\log}$ if and only if G has a clique of size k. However, $\text{Clique}_{\log}$ can be solved in time $|V(G)|^{\mathcal{O}(\log_2 |V(G)|)}$ by brute force, as the answer is trivial if $k > \log_2 |V(G)|$. That is, the problem can be solved in quasi-polynomial time $n^{\mathcal{O}(\log n)}$, which we do not expect to be possible for NP-hard problems.

In general, parameterized reductions and polynomial-time reductions are incomparable. However, the vast majority of parameterized reductions (including most reductions appearing in this chapter) are actually polynomial-time reductions.

Later in this section, a hardness proof for Dominating Set on Tournaments will be a more natural example of a parameterized reduction with running time exponentially depending on the parameter k.

13.2 Problems at least as hard as Clique

In this section, we present a series of parameterized reductions showing that Clique can be reduced to various basic problems. This means that these problems are at least as hard as Clique in the sense that if any of them is FPT, then it follows that Clique is FPT as well. This is a strong evidence that none of the problems considered in this section is FPT.

To be more precise, the reductions given in this section are not all from Clique. We use the fact, stated in the following theorem, that parameterized reductions compose: that is, a reduction from A to B and reduction from B to C implies that there is a reduction from A to C. Therefore, to show that there is a parameterized reduction from Clique to a problem C, it is sufficient to reduce a problem B to C, where B is a problem to which we already have a reduction from Clique.

Theorem 13.3. *If there are parameterized reductions from A to B and from B to C, then there is a parameterized reduction from A to C.*

Proof. Let $\mathcal{R}_1$ be a parameterized reduction from A to B (with running time $f_1(k) \cdot |x|^{c_1}$ and parameter bound $g_1(k)$) and let $\mathcal{R}_2$ be a parameterized reduction from B to C (with running time $f_2(k) \cdot |x|^{c_2}$ and parameter bound $g_2(k)$). As discussed after Definition 13.1, we may assume that the functions f_1, f_2, g_1, g_2 are nondecreasing. Given an instance (x, k) of A, reduction $\mathcal{R}$ first uses $\mathcal{R}_1$ to create an equivalent instance (x', k') of B (with $|x'| \leq f_1(k) \cdot |x|^{c_1}$ and $k' \leq g_1(k)$), and then uses reduction $\mathcal{R}_2$ on (x', k') to create an equivalent instance of C (with $k'' \leq g_2(k')$). It is clear that (x, k) is

a yes-instance of A if and only if (x'', k'') is a yes-instance of C. We have $k'' \leq g_2(k') \leq g_2(g_1(k))$ (using that g_2 is nondecreasing), thus k'' is bounded by a computable function of k. The running time of the reduction is

$$\begin{aligned} f_1(k) \cdot |x|^{c_1} + f_2(k') \cdot |x'|^{c_2} &\leq f_1(k) \cdot |x|^{c_1} + f_2(g_1(k)) \cdot (f_1(k) \cdot |x|^{c_1})^{c_2} \\ &\leq (f_1(k) + f_2(g_1(k)) f_1(k)) \cdot |x|^{c_1 c_2} \\ &= f^*(k) \cdot |x|^{c_1 c_2}, \end{aligned}$$

where $f^*(k) = f_1(k) + f_2(g_1(k)) f_1(k)$ is a computable function (we use in the first inequality that f_2 is nondecreasing). Thus reduction $\mathcal{R}$ satisfies all requirements of Definition 13.1. □

It is common practice to prove NP-hardness for the most restricted version of a problem, as this can be convenient when the problem becomes the source of a reduction later: it can be easier to devise a reduction if input instances of only some restricted form need to be considered. The same principle holds for parameterized problems and in particular for Clique, which is the source of most parameterized reductions. We show first that Clique remains hard even if the graph is regular, that is, every vertex has the same degree.

Theorem 13.4. *There is a parameterized reduction from* Clique *to* Clique *on regular graphs.*

Proof. Given an instance (G, k) of Clique, we create a graph G' as described below and output the instance (G', k). If $k \leq 2$, then the Clique problem is trivial, hence we can output a trivial yes- or no-instance. Let d be the maximum degree of G.

(i) Take d distinct copies $G_1, \ldots, G_d$ of G and let v_i be the copy of $v \in V(G)$ in graph G_i.
(ii) For every vertex $v \in V(G)$, let us introduce a set V_v of $d - d_G(v)$ vertices and add edges between every vertex of V_v and every v_i for $1 \leq i \leq d$.

Observe that every vertex of G' has degree exactly d. To prove the correctness of the reduction, we claim that G has a k-clique if and only if G' has. The left to right implication is clear: copies of G appear as subgraphs in G', thus any clique in G gives a corresponding clique in G'. For the reverse direction, observe that the vertices introduced in step (ii) do not appear in any triangles. Therefore, assuming $k \geq 3$, these vertices cannot be part of a k-clique. Removing these vertices gives d disjoint copies of G, thus any k-clique appearing there implies the existence of a k-clique in G. □

As the complement of a regular graph is also regular, we get an analogous result for Independent Set.

Proposition 13.5. *There is a parameterized reduction from* Clique *to* Independent Set *on regular graphs.*

Note that we have restricted the graph to be regular, but we did not specify that the graph is, say, 3-regular or has any other fixed degree bound. Indeed, Clique and Independent Set are both FPT on r-regular graphs for every fixed integer r, and moreover, FPT with combined parameters $k + r$ (see Exercise 3.2). This has to be contrasted with the fact that Independent Set is NP-hard on 3-regular graphs.

The following reduction shows an example where we can exploit that the graph appearing in the Clique instance is regular. The input to the Partial Vertex Cover problem is a graph G with two integers k and s, and (G, k, s) is a yes-instance if G has a set S of k vertices that covers at least s edges (we say that a set S covers an edge xy if at least one of x and y is in S). Vertex Cover is the special case $s = |E(G)|$; the following reduction shows that the general Partial Vertex Cover problem is probably harder than Vertex Cover.

Theorem 13.6. *There is a parameterized reduction from* Independent Set *on regular graphs to* Partial Vertex Cover *parameterized by* k.

Proof. Let (G, k) be an instance of Independent Set, where G is an r-regular graph. We claim that G has an independent set of size k if and only if (G, k, rk) is a yes-instance of Partial Vertex Cover. If G has an independent set S of size k, then every edge of G is covered by at most one vertex of S, hence together they cover exactly rk edges. Conversely, suppose that G has a set S of k vertices covering rk edges. As each vertex of G covers exactly r edges, this is only possible if no edge of G is covered twice by these k vertices, that is, if they form an independent set. □

Note that Partial Vertex Cover is fixed-parameter tractable parameterized by s, the number of edges to cover (Exercise 5.11).

Next we consider a version of the Clique problem that is a useful starting point for hardness proofs. The input of Multicolored Clique[2] (also called the Partitioned Clique) consists of a graph G, an integer k, and a partition $(V_1, \ldots, V_k)$ of the vertices of G; the task is to decide if there is a k-clique containing exactly one vertex from each set V_i.

Theorem 13.7. *There is a parameterized reduction from* Clique *on regular graphs to* Multicolored Clique *on regular graphs.*

Proof. Let (G, k) be an instance of Clique, where G is an r-regular graph on n vertices. We construct an instance $(G', k, (V_1, \ldots, V_k))$ as follows. For every vertex $v \in V(G)$, we introduce k vertices $v_1, \ldots, v_k$ into G'. The set V_i contains v_i for every $v \in V(G)$. We define the edges of G' as follows: :

(i) Every V_i is a set of n independent vertices.

[2] The name "multicolored" comes from an alternate way of defining the problem: we can say that the vertices of G are colored by k colors and we are looking for a clique where every vertex has a distinct color.

(ii) If u and v are different and adjacent in G, then we make u_i and v_j adjacent for every $1 \leq i, j \leq k$, $i \neq j$.

Observe that the degree of every vertex in G' is $(k-1)r$: vertex v_i is adjacent to r vertices from each V_j with $i \neq j$.

We claim that G has a k-clique if and only if G' has a k-clique with exactly one vertex from each V_i. Suppose that $v^1, \ldots, v^k$ are the vertices of a k-clique in G (ordered arbitrarily). Then $v_1^1 \in V_1, \ldots, v_k^k \in V_k$ is a k-clique in G'. Conversely, suppose that $v_1^1 \in V_1, \ldots, v_k^k \in V_k$ is a k-clique in G'. It follows from the definition of G' that v_i^i and v_j^j are only adjacent in G' if v^i and v^j are different and adjacent in G. Therefore, $\{v^1, \ldots, v^k\}$ induces a k-clique in G. □

One may analogously define the multicolored version of Independent Set, where the k vertices of the independent set has to contain one vertex from each class of the given partition $(V_1, \ldots, V_k)$. Taking the complement of the graph is a parameterized reduction between Clique and Independent Set. Observe that this also holds for the multicolored version.

Corollary 13.8. *There are parameterized reductions between* Multicolored Clique *on regular graphs and* Multicolored Independent Set *on regular graphs.*

Our next reduction gives evidence for the fixed-parameter intractability of Dominating Set.

Theorem 13.9. *There is a parameterized reduction from* Multicolored Independent Set *to* Dominating Set.

Proof. Let $(G, k, (V_1, \ldots, V_k))$ be an instance of Multicolored Independent Set. We construct a graph G' the following way (see Fig. 13.1).

(i) For every vertex $v \in V(G)$, we introduce v into G'.
(ii) For every $1 \leq i \leq k$, we make the set V_i a clique in G'.
(iii) For every $1 \leq i \leq k$, we introduce two new vertices x_i, y_i into G' and make them adjacent to every vertex of V_i.
(iv) For every edge $e \in E(G)$ with endpoints $u \in V_i$ and $v \in V_j$, we introduce a vertex w_e into G' and make it adjacent to every vertex of $(V_i \cup V_j) \setminus \{u, v\}$.

The intuitive idea of the reduction is the following. The vertices x_i and y_i ensure that a dominating set of size k has to select exactly one vertex from each V_i. The vertices w_e ensure that the selected vertices form an independent set in the original graph: if both endpoints of an edge e are selected, then the vertex w_e is not dominated.

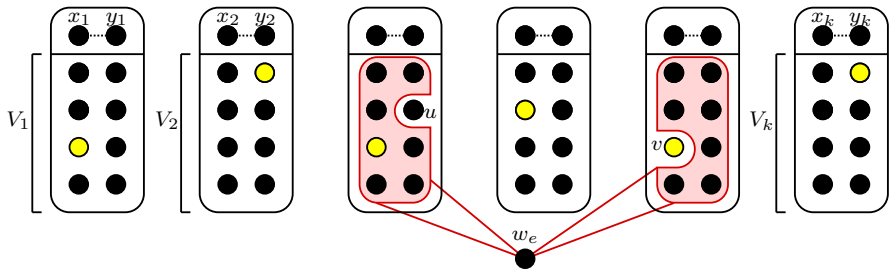


Fig. 13.1: The reduction from Multicolored Independent Set to Dominating Set in Theorem 13.9. The set $V_i \cup \{x_i, y_i\}$ induces a clique minus the edge $x_i y_i$. If edge e connectes $u \in V_i$ and $v \in V_j$, then we introduce a vertex w_e adjacent to $(V_i \cup V_j) \setminus \{u, v\}$.

Formally, we claim that G has an independent set with exactly one vertex from each V_i if and only if G' has a dominating set of size k. Suppose that there is an independent set I in G with one vertex from each V_i; we show that it is a dominating set in G'. First, as $I \cap V_i \neq \emptyset$, the set I dominates the set $V_i \cup \{x_i, y_i\}$. Consider now a vertex w_e, where $u \in V_i$ and $v \in V_j$ are the endpoints of edge $e \in E(G)$. As u and v are adjacent, at least one of them is not in I. In other words, I contains a vertex of either $V_i \setminus \{u\}$ or $V_j \setminus \{v\}$, that is, I contains a vertex of $(V_i \cup V_j) \setminus \{u, v\}$, which dominates w_e.

To prove the reverse direction of the equivalence, suppose now that D is a dominating set of size k in G'. To dominate vertices x_i and y_i, the set D has to contain at least one vertex from $V_i \cup \{x_i, y_i\}$. As these sets are disjoint for different values of i, we can assume that D contains exactly one vertex from each of $V_i \cup \{x_i, y_i\}$. As x_i and y_i are not adjacent, this vertex of D cannot be x_i or y_i. Therefore, let us assume that the dominating set D consists of the vertices $v_1 \in V_1, \ldots, v_k \in V_k$. We claim that D is an independent set in G. Suppose that v_i and v_j are the endpoints of an edge e. Then vertex w_e of G' is adjacent only to $(V_i \cup V_j) \setminus \{v_i, v_j\}$, and hence D does not dominate w_e, a contradiction. □

As there is a very simple reduction between (multicolored versions of) Clique and Independent Set, a reduction from one can be reinterpreted to be a reduction from the other with minimal changes. However, sometimes choosing one of the two problems is more convenient notationally or conceptually; for example, in the proof of Theorem 13.9, it is convenient notationally that (iv) adds a new vertex w_e for every edge e (rather than for every nonedge).

Given a set system $\mathcal{F}$ over a universe U and an integer k, the Set Cover problem asks if there are k sets in $\mathcal{F}$ such that their union is U. In this chapter, we always parameterize Set Cover by the number k of selected sets in the solution. The following easy reduction shows that Set Cover

is as hard as Dominating Set (Exercise 13.2 asks for a reduction in the reverse direction).

Theorem 13.10. *There is a parameterized reduction from* Dominating Set *to* Set Cover.

Proof. Let (G, k) be an instance of Dominating Set. We create an instance $(\mathcal{F}, U, k)$ of Set Cover as follows. We let $U := V(G)$ and for every $v \in V(G)$, we introduce the set $N_G[v]$ (the closed neighborhood of v) into $\mathcal{F}$. Suppose that D is a dominating of size k in G. Then the union of the corresponding k sets of $\mathcal{F}$ covers U: an uncovered element would correspond to a vertex of G not dominated by D. Conversely, if the union of k sets in $\mathcal{F}$ is U, then the corresponding k vertices of G dominate every vertex: a vertex not dominated in G would correspond to an element of U not covered by the k sets. □

On directed graphs, we define dominating set in the following way: a set $D \subseteq V(G)$ is a dominating set of G if for every vertex $v \in V(G) \setminus D$, there is a vertex $u \in D$ such that edge (u, v) is in G. Equivalently, we can say that we want the union of the closed outneighborhoods to cover every vertex. It is easy to reduce Dominating Set on undirected graphs to Dominating Set on directed graphs: we can replace each undirected edge xy with two directed edges (x, y) and (y, x) (in other words, we replace xy with a bidirected edge between x and y). Furthermore, the reduction from Dominating Set to Set Cover appearing in Theorem 13.10 can be adapted to directed graphs. Therefore, Dominating Set on undirected graphs, Dominating Set on directed graphs, and Set Cover are reducible to each other.

A very interesting special case of Dominating Set on directed graphs is when the input graph is a *tournament,* that is, for every pair u, v of distinct vertices, exactly one of (u, v) and (v, u) is in the graph. It is easy to see that every tournament on n vertices has a dominating set of size $\mathcal{O}(\log n)$.

Lemma 13.11. *Every tournament on n vertices has a dominating set of size at most $1 + \log n$.*

Proof. We prove the claim by induction on n. For $n = 1$ it is trivial, so let us assume $n \geq 2$.

Since T has $\binom{n}{2}$ edges, the sum of outdegrees of all the vertices is equal to $\binom{n}{2} = \frac{n(n-1)}{2}$. As there are n vertices in T, there exists some vertex u with outdegree at least $\frac{n-1}{2}$. Let U^+ be the set of outneighbors of u, and let U^- be the set of inneighbors of u. Then U^+ and U^- form a partition of $V(T) \setminus \{u\}$, and $|U^-| \leq \frac{n-1}{2} \leq \lfloor \frac{n}{2} \rfloor$. Since $\lfloor \frac{n}{2} \rfloor < n$, by induction hypothesis we may find a set $D' \subseteq U^-$ that is a dominating set in $T[U^-]$ and has at most $1 + \log \lfloor \frac{n}{2} \rfloor \leq \log n$ vertices. Then $D := D' \cup \{u\}$ is a dominating set in T, and has size at most $1 + \log n$. □

Therefore, Dominating Set on Tournaments can be solved by brute force in time $n^{\mathcal{O}(\log n)}$, making it very unlikely that it is NP-hard. Is there

such a spectacular collapse of complexity compared to general directed graphs also for the parameterized version of the problem? Surprisingly, we are able to show that this is not true: there is a parameterized reduction from Set Cover to Dominating Set on Tournaments.

The crucial building block in our reduction will be a construction of sufficiently small tournaments that do not admit a dominating set of size k. Such tournaments are traditionally called *k-paradoxical*, and were studied intensively in combinatorics. Note that by Lemma 13.11, a k-paradoxical tournament must have at least 2^k vertices. It turns out that once we increase the size of the vertex set slightly over this threshold, the k-paradoxical tournaments start to appear.

Theorem 13.12. *For every $n \geq r_k = 2 \cdot 2^k \cdot k^2$ there exists a tournament on n vertices that does not admit a dominating set of size k.*

The proof of Theorem 13.12, due to Erdős, is a classic application of the probabilistic method: a randomly sampled tournament on at least r_k vertices does not admit a dominating set of size k with positive probability. The reader is asked to verify this fact in Exercise 13.4. We remark that the multiplicative constant 2 in the function r_k can be in fact substituted with $\ln 2 + o_k(1)$.

In our reduction we need to *construct* a k-paradoxical tournament, and hence it seems necessary to have a constructive, deterministic version of Theorem 13.12. Apparently, there exist deterministic constructions that achieve similar asymptotic size of the vertex set as Theorem 13.12; in Exercise 13.5, the reader is asked to work out details of a simple deterministic construction with slightly worse bounds. We can, however, circumvent the need of a deterministic construction at the cost of allowing the construction to run in doubly exponential time in terms of k. The running time will be still fixed-parameter tractable, which is sufficient for our purposes.

Lemma 13.13. *There exists an algorithm that, given an integer k, works in time $2^{\binom{r_k}{2}} \cdot r_k^{k+1} \cdot k^{\mathcal{O}(1)}$ and outputs a tournament T_k that has r_k vertices and does not admit a dominating set of size k.*

Proof. We construct T_k by iterating through all the $2^{\binom{r_k}{2}}$ tournaments on r_k vertices, and for each of them verifying in time $r_k^{k+1} \cdot k^{\mathcal{O}(1)}$ whether it is k-paradoxical by checking all the k-tuples of vertices. Theorem 13.12 guarantees that one of the verified tournaments will be in fact k-paradoxical. □

We are finally ready to provide the reduction.

Theorem 13.14. *There is a parameterized reduction from* Set Cover *to* Dominating Set on Tournaments.

Proof. The reduction starts with an instance $(\mathcal{F}, U, k)$ of Set Cover, and outputs an equivalent instance $(T, k+1)$ of Dominating Set on Tournaments. The first step is a construction of a $(k+1)$-paradoxical tournament

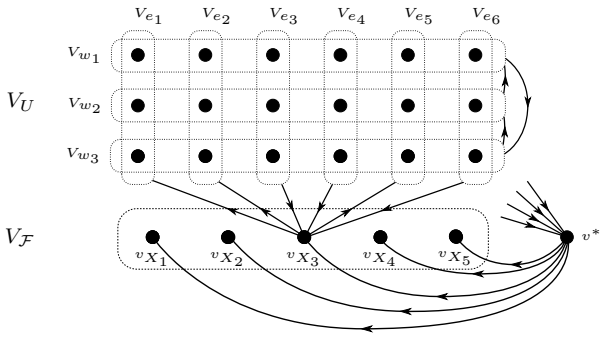


Fig. 13.2: A schematic view of the construction of Theorem 13.14 for a universe of size 6, family of size 5, and S being a cyclic tournament on 3 vertices. Edges adjacent to the sets V_{e_i} and V_{w_j} represent appropriately oriented complete bipartite graphs, and edges created in (i) between $V_{\mathcal{F}}$ and V_U have been depicted only for the set $X_3 = \{e_1, e_2, e_5\}$.

$S = T_{k+1}$ on r_{k+1} vertices using Lemma 13.13; this contributes with factor $2^{\binom{r(k+1)}{2}} \cdot r(k+1)^{k+2} \cdot (k+1)^{\mathcal{O}(1)}$ to the running time, which is just a function of k.

The vertex set of the constructed tournament T is defined as follows:

(i) For every $e \in U$, create a set of r_{k+1} vertices $V_e = \{v_{e,w} \ : \ w \in V(S)\}$, one for each vertex of S. Let $V_w = \{v_{e,w} \ : \ e \in U\}$, and let $V_U = \bigcup_{e \in U} V_e = \bigcup_{w \in V(S)} V_w$.
(ii) For every $X \in \mathcal{F}$, create one vertex v_X. Let $V_{\mathcal{F}} = \{v_X \ : \ X \in \mathcal{F}\}$.
(iii) Moreover, create one vertex v^*.

We now create the edge set of T.

(i) For every set $X \in \mathcal{F}$ and every element $e \in U$, if $e \in X$ then introduce an edge from v_X to every vertex of V_e, and if $e \notin X$ then introduce an edge from every vertex of V_e to v_X.
(ii) For every set $X \in \mathcal{F}$, introduce an edge (v^*, v_X).
(iii) For every element $e \in X$ and $w \in V(S)$, introduce an edge $(v_{e,w}, v^*)$.
(iv) For every $w_1, w_2 \in V(S)$ with $w_1 \neq w_2$, introduce an edge from every vertex of V_{w_1} to every vertex of V_{w_2} if $(w_1, w_2) \in E(S)$, and introduce the reverse edges if $(w_2, w_1) \in E(S)$.
(v) For every $w \in V(S)$, put edges between vertices of V_w arbitrarily.
(vi) Finally, put the edges between vertices of $V_{\mathcal{F}}$ arbitrarily.

It is easy to see that the constructed digraph T is indeed a tournament, and that the construction can be performed in time polynomial in $|U|$, $|\mathcal{F}|$, and r_{k+1} (provided S is already constructed). We now claim that T admits a dominating set of size $k+1$ if and only if the input instance $(\mathcal{F}, U, k)$ is a yes-instance.

Assume first that $\mathcal{G} \subseteq \mathcal{F}$ is a subfamily of size at most k such that $\bigcup \mathcal{G} = U$. Consider $D = \{v^*\} \cup \{v_X \ : \ X \in \mathcal{G}\}$. Clearly $|D| \leq k+1$, and observe that D is a dominating set of T: each vertex of $V_{\mathcal{F}}$ is dominated by v^*, while each vertex $v_{e,w} \in V_U$ is dominated by a vertex $v_X \in D$ for $X \in \mathcal{G}$ such that $e \in X$.

Conversely, suppose that T admits a dominating set D such that $|D| \leq k+1$. Since D has to dominate v^*, then either D contains v^* or at least one vertex of V_U. Consequently, $|D \cap V_{\mathcal{F}}| \leq k$. Let $\mathcal{G} = \{X \in \mathcal{F} \ : \ v_X \in D\}$. Clearly $|\mathcal{G}| \leq k$, so it suffices to prove that $\bigcup \mathcal{G} = U$.

For the sake of contradiction assume that there exists some $e_0 \in U$ that does not belong to any set of $\mathcal{G}$. Let $Z \subseteq V(S)$ be the set of all vertices $z \in V(S)$ such that $D \cap V_z \neq \emptyset$. Since $|D| \leq k+1$, we also have $|Z| \leq k+1$. Since S is $(k+1)$-paradoxical, we have that there exists some vertex $w_0 \in V(S)$ that is not dominated by Z in S. Consider now the vertex v_{e_0,w_0} in T. By the definition of the edges between $V_{\mathcal{F}}$ and V_U, introduced in (i), we have that v_{e_0,w_0} is not dominated by $D \cap V_{\mathcal{F}}$, since then e_0 would belong to $\bigcup \mathcal{G}$ by the definition of $\mathcal{G}$. Since Z does not dominate w_0 in S, then by the definition of the edges introduced in (iv) it also follows that $D \cap V_U$ does not dominate v_{e_0,w_0}. Note that in particular $w_0 \notin Z$, so $D \cap V_{w_0} = \emptyset$ and v_{e_0,w_0} cannot be dominated from within V_{w_0}. Finally, v_{e_0,w_0} cannot be dominated by $D \cap \{v^*\}$, since v^* does not have any outneighbor in V_U. We infer that v_{e_0,w_0} is not dominated by D at all, which contradicts the assumption that D is a dominating set in T. □

Note that, unlike most parameterized reductions in this book and also in the literature, the reduction in Theorem 13.14 is *not* a polynomial-time reduction: the size of the constructed instance has exponential dependence on k and the running time depends double exponentially on k. Therefore, this reduction crucially uses the fact that property 3 in Definition 13.1 allows $f(k) \cdot |x|^{\mathcal{O}(1)}$ time instead of just $|x|^{\mathcal{O}(1)}$.

Connected Dominating Set is the variant of Dominating Set where we additionally require that the dominating set induces a connected graph. Not surprisingly, this version of the problem is also hard.

Theorem 13.15. *There is a parameterized reduction from* Dominating Set *to* Connected Dominating Set.

Proof. Let (G, k) be an instance of Dominating Set. We construct a graph G' the following way.

(i) For every vertex $v \in V(G)$, we introduce two adjacent vertices v_1, v_2.
(ii) We make the set $\{v_1 \ : \ v \in V(G)\}$ a clique K of size $|V(G)|$.
(iii) We make v_1 and u_2 adjacent if v and u are adjacent in G.

We claim that (G, k) is a yes-instance of Dominating Set if and only if (G', k) is a yes-instance of Connected Dominating Set. Suppose first that $S = \{v^1, \ldots, v^k\}$ is a dominating set of size k in G. Then we claim that

$S' = \{v_1^1, \ldots, v_1^k\}$ is a connected dominating set of size k in G'. Clearly, $G'[S']$ is a clique and hence it is connected. To see that S' is a dominating set in G', observe that v_1^1 dominates K, and if u is dominated by v^i in G, then u_2 is dominated by v_1^i in G'.

For the proof of the reverse direction of the equivalence, let S' be a connected dominating set of size k in G'. Let v be in S if at least one of v_1 and v_2 is in S'; clearly, $|S| \leq |S'| = k$. We claim that S is a dominating set of G. Consider any vertex $u \in V(G)$. Vertex u_2 of G' is dominated by some vertex v_1 or v_2 that belongs to S'. Then v is in S and, by the construction of G', it dominates u in G, as required. □

13.3 The W-hierarchy

As we have discussed above, most of the natural NP-hard problems are equivalent to each other (as they are NP-complete), but this does not seem to be true for parameterized problems. For example, we have shown that (Multicolored) Independent Set can be reduced to Dominating Set (Theorem 13.9), but it is not known if there is a parameterized reduction in the other direction. This suggests that, unlike in the case of NP-complete problems, there is a hierarchy of hard parameterized problems, with, e.g., Independent Set and Dominating Set occupying different levels of this hierarchy. Downey and Fellows introduced the W-hierarchy in an attempt to capture the exact complexity of various hard parameterized problems. In this section, we briefly overview the fundamental definitions and results related to this hierarchy. It has to be noted, however, that if our only concern is to give some evidence that a specific problem is not fixed-parameter tractable (which is the usual viewpoint of an algorithm designer), then the exact structure of this hierarchy is irrelevant: the reductions from Clique, as presented in Section 13.2 and appearing later in this chapter, already give a practical evidence that a problem is unlikely to be FPT. Nevertheless, the reader should be aware of the basic terminology of the W-hierarchy, as the parameterized algorithms and complexity literature usually assumes its knowledge.

A *Boolean circuit* is a directed acyclic graph where the nodes are labeled the following way:

- every node of indegree 0 is an *input node,*
- every node of indegree 1 is a *negation node,*
- every node of indegree ≥ 2 is either an *and-node* or an *or-node.*

Additional, exactly one of the nodes with outdegree 0 is labeled as the *output node* (in addition to being, e.g., an and-node). The *depth* of the

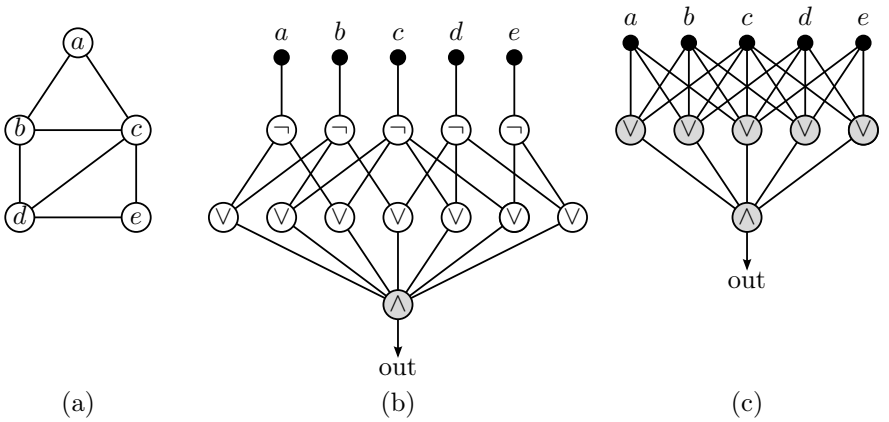


Fig. 13.3: (a) A graph G with 5 vertices and 7 edges. (b) A depth-3, weft-1 circuit satisfied by the independent sets of G. Each or-node corresponds to an edge of G and has indegree 2. (c) A depth-2, weft-2 circuit satisfied by the dominating sets of G. Each or-node corresponds to a vertex of G and its inneighbors correspond to the closed neighborhood to the vertex.

circuit is the maximum length of a path from an input node to the output node.

Assigning 0-1 values to the input nodes determines the value of every node in the obvious way. In particular, if the value of the output gate is 1 in an assignment to the input nodes, then we say that the assignment *satisfies* the circuit. It can be easily checked in polynomial time if a given assignment satisfies the circuit.

Deciding if a circuit has a satisfying assignment is clearly an NP-complete problem: for example, 3-SAT is its special case. We can define a parameterized version of finding a satisfying assignment in the following way. The *weight* of an assignment is the number of input gates receiving value 1. In the Weighted Circuit Satisfiability (WCS) problem, we are given a circuit C and an integer k, the task is to decide if C has a satisfying assignment of weight *exactly* k. By trying all the $\mathcal{O}(n^k)$ assignments of weight exactly k and then checking whether it satisfies C, we can solve the problem in polynomial time for every fixed k. However, the problem does not seem to be fixed-parameter tractable: it is quite easy to reduce hard parameterized problems such as Clique or Dominating Set to Weighted Circuit Satisfiability (see Fig. 13.3).

The levels of the W-hierarchy are defined by restricting Weighted Circuit Satisfiability to various classes of circuits. Formally, if $\mathcal{C}$ is a class of circuits, then we define WCS[$\mathcal{C}$] to be the restriction of the problem where

the input circuit C belongs to $\mathcal{C}$. It seems to be a natural idea to restrict the depth of the circuits in the class $\mathcal{C}$; there is a large literature on the expressive power of bounded-depth circuits. However, the restrictions we need here is a bit more subtle than that. First, we distinguish *small nodes*, which have indegree at most 2, and *large nodes*, which have indegree > 2 (as we shall see later, the choice of the constant 2 is not essential here). The *weft*[3] of a circuit is the maximum number of large nodes on a path from an input node to the output node. We denote by $\mathcal{C}_{t,d}$ the class of circuits with weft at most t and depth at most d.

Definition 13.16 (W-hierarchy). For $t \geq 1$, a parameterized problem P belongs to the class $\mathrm{W}[t]$, if there is a parameterized reduction from P to $\mathrm{WCS}[\mathcal{C}_{t,d}]$ for some $d \geq 1$.

Exercise 13.6 shows that the constant 2 in the definition of small/large nodes is not essential: any other constant ≥ 2 would result in the same definition for $\mathrm{W}[t]$.

As indicated in Fig. 13.3, it is possible to reduce Independent Set to Weighted Circuit Satisfiability for circuits of depth 3 and weft 1. By Definition 13.16, membership in W[1] follows.

Proposition 13.17. Independent Set *is in* W[1].

One can also show the converse statement: every problem in W[1] can be reduced to Independent Set, that is, the problem is W[1]-hard. The proof of this statement is nontrivial and beyond the scope of this brief introduction. Together with Proposition 13.17, it follows that Independent Set is W[1]-complete.

Theorem 13.18 ([151]). Independent Set *is* W[1]*-complete.*

The circuit in Fig. 13.3 expressing Independent Set has a very specific form: it consists of one layer of negation nodes, one layer of small or-nodes with indegree 2, and a final layer consisting of a single large and-node. Theorem 13.18 shows that this form is in some sense canonical for weft-1 circuits: the satisfiability problem for (bounded-depth) weft-1 circuits can be reduced to weft-1 circuits of this form.

What would be the corresponding canonical forms for weft-t circuits for $t \geq 2$? We need some definitions first. We say that a Boolean formula is t-*normalized* if it is of the following form: it is the conjunction of disjunctions of conjunctions of disjunctions, ..., alternating for t levels. For example, the formula

$$(x_1 \vee \bar{x}_3 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4 \vee x_5) \wedge (x_1 \vee \bar{x}_2)$$

is 2-normalized, as it is the conjunction of disjunctions of literals. More formally, we first define both Δ_0 and Γ_0 to be the set of formulas consisting of

[3] Weft is a term related to weaving cloth: it is the thread that runs from side to side in the fabric.

only a single literal. Then for $t \geq 1$, we define Δ_t to contain formulas that are the disjunction of an arbitrary number of Γ_{t-1} formulas and Γ_t to contain formulas that are the conjunction of an arbitrary number of Δ_{t-1} formulas. With these definitions, Γ_t is exactly the set of t-normalized formulas. We say that a t-normalized formula is *monotone* (resp., *antimonotone*) if every literal is positive (resp., negated).

The Weighted t-normalized Satisfiability problem is the weighted satisfiability problem corresponding to circuits representing t-normalized formulas; the monotone/antimonotone versions of the problem are defined in the obvious way. The following theorem, whose proof again goes beyond the scope of this introduction, characterizes the classes W[t] for $t \geq 2$ in terms of these problems.

Theorem 13.19 ([149, 150, 151]).

1. *For every even $t \geq 2$, the following problems are* W[t]*-complete:*
   - Weighted t-normalized Satisfiability
   - Weighted Monotone t-normalized Satisfiability
   - Weighted Monotone $(t+1)$-normalized Satisfiability
2. *For every odd $t \geq 3$, the following problems are* W[t]*-complete:*
   - Weighted t-normalized Satisfiability
   - Weighted Antimonotone t-normalized Satisfiability
   - Weighted Antimonotone $(t+1)$-normalized Satisfiability

Theorem 13.19 has three rather surprising aspects. First, it shows that t-normalized formulas are sufficient to express the full power of (bounded-depth) weft-t circuits. These formulas have very specific structures: besides the negation nodes at the inputs, they consist of t layers of large nodes, alternating between and-nodes and or-nodes. One can interpret Theorem 13.19 as saying that additional layers of small nodes do not increase the expressive power, as t-normalized formulas already give rise to W[t]-complete problems. This justifies why we defined W[t] based on weft instead of depth. Second, Theorem 13.19 states that already monotone t-normalized formulas (for even t) and antimonotone t-normalized formulas (for odd t) are sufficient to capture the full power of W[t]. Finally, Theorem 13.19 also states the surprising fact that the power of monotone/antimonotone t-normalized and $(t+1)$-normalized formulas coincide for certain values of t.

Let us focus now on W[2], the second level of the hierarchy. Fig. 13.3 shows that it is possible to reduce Dominating Set to Weighted Circuit Satisfiability for circuits of depth 2 and weft 2, hence membership in W[2] follows. We have seen in Section 13.2, Dominating Set and Set Cover (and hence Hitting Set) are equally hard with respect to parameterized reductions (Theorem 13.10 and Exercise 13.2).

Proposition 13.20. Dominating Set, Set Cover, Hitting Set *are in* W[2].

We observe now that these problems are complete for W[2]. All we have to note is that Hitting Set is equivalent to Weighted Monotone 2-normalized Satisfiability. A 2-monotone normalized formula is of the form

$$(x_1 \vee x_3 \vee x_6) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_5 \vee x_6).$$

To satisfy such a formula, we have to set at least one variable x_i to 1 in each clause. This is exactly the Hitting Set problem, where the universe is the set of variables and the sets are the clauses. Thus Theorem 13.19 shows that Hitting Set is W[2]-complete.

Theorem 13.21. Dominating Set, Set Cover, *and* Hitting Set *are* W[2]*-complete.*

The usual expectation is that the W-hierarchy is a proper hierarchy: $\mathrm{W}[t] \neq \mathrm{W}[t+1]$ for every $t \geq 1$. In Theorem 13.9, we have seen a reduction from Independent Set to Dominating Set. In light of Theorem 13.21, we do not expect a reduction in the reverse direction: this would imply that Independent Set is both W[1]- and W[2]-complete and hence $\mathrm{W}[1] = \mathrm{W}[2]$ would follow.

*13.4 Turing machines

Clique and Independent Set are among the most studied optimization problems in computer science, hence the fact that no algorithm with running time $n^{o(k)}$ was found despite massive efforts is a good practical evidence suggesting that these problems are not fixed-parameter tractable. The reductions in Section 13.2 (and those appearing later in Section 13.6) transfer this apparent intractability to several other problems. In this section, we give a more theoretical supporting evidence for the intractability of these problems.

Turing machines formalize the intuitive notion of computation in a precise mathematical way. Computational problems where the task is to predict what a Turing machine will do are typically very hard: as Turing machines can express arbitrarily complex computations, it seems that one needs to simulate the computation of the Turing machine to answer these questions. In the case of the Halting problem, there is even a formal proof of hardness: a classic diagonalization argument shows that there is no algorithm deciding whether a Turing machine halts in a finite number of steps. The $\mathrm{P} \neq \mathrm{NP}$ problem is a time-bounded version of this question: one has to decide in deterministic polynomial time whether a nondeterministic polynomial-time Turing machine has an accepting path. While there is no proof that this is not possible, it seems that a nondeterministic Turing machine is such an opaque and generic object that it is unlikely that one can avoid simulating a superpolynomial number of computation paths.

We define a parameterized problem analogous to the acceptance problem of nondeterministic polynomial-time Turing machines, and use it to give further evidence that CLIQUE is not fixed-parameter tractable. In the SHORT TURING MACHINE ACCEPTANCE problem, the input contains the description of a nondeterministic Turing machine M, a string x, and an integer k; the task is to decide if M with input x has an computation path reaching an accepting state in at most k steps. If the reader sees this problem for the first time, it might appear that simulating a Turing machine for a small number k of steps is an easy task, but let us point out that a nondeterministic Turing machine may branch into an unbounded number of states (bounded only by the length n of the input), hence naive simulation of k steps would lead to an $n^{\mathcal{O}(k)}$ time algorithm. Therefore, the same intuitive argument suggesting P $\neq$ NP also supports the conjecture that SHORT TURING MACHINE ACCEPTANCE is not fixed-parameter tractable: it is unlikely that a problem that requires understanding a Turing machine with $n^{\mathcal{O}(k)}$ computation paths is FPT.

We present a parameterized reduction from SHORT TURING MACHINE ACCEPTANCE to INDEPENDENT SET in this section. This can be considered as evidence for the fact that INDEPENDENT SET (as well as all the other problems considered in Section 13.2) are not fixed-parameter tractable. The reduction is fairly tedious, but the principle is simple. An accepting computation path of k steps can be fully described by $\mathcal{O}(k^2)$ pieces of information: a sequence of k states, a sequence of k transitions, the content of the first k cells of the tape of a Turing machine in each of the first k steps. We need to construct a graph where the choice of vertices for an independent set of size $\mathcal{O}(k^2)$ represents all this information, and then make sure that this information is consistent, that is, correctly describes a sequence of configurations forming an accepting computation path.

We assume that the reader is familiar with the concept of Turing machines (see standard textbooks such as [268, 416, 379] for more background). Here we review only the basic notation required for the discussion of Turing machines. Since we are using Turing machines as an evidence for hardness, and not as a basis for building complexity theory, we avoid treating them in greater detail. We remind the reader that certain details and conventions in the definition (such as whether the tape is infinite in both directions, whether the machine can leave an accepting state, etc.) usually do not matter for the computational power of Turing machines and proofs can be changed accordingly.

A *single-tape nondeterministic Turing machine* is described by a tuple $M = (Q, \Sigma, \Delta, s_0, F)$, where

- Q is a finite set of states,
- Σ is the alphabet,
- $s_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states, and

- $\Delta \subseteq Q \times (\Sigma \cup \{\$, \square\}) \times Q \times (\Sigma \cup \{\$\}) \times \{\leftarrow, -, \rightarrow\}$ is the transition relation.

The meaning of the transition relation is the following: if, for example, $(q, a, q', b, \leftarrow) \in \Delta$, then this means that whenever the Turing machine is in state q and symbol a appears under the head, then one possible legal step is to move into state q', write b on the tape, and move the head one step to the left. The special symbol $\square$ represents empty cells of the tape and special symbol \$ appears at the left end (position 0) of the tape. Initially, the tape contains an input word x followed by an infinite sequence of $\square$ symbols; the head is at position 1. We assume that whenever the Turing machine reads the symbol \$, then it does not change it (that is, writes \$ on it) and does not move further to the left.

Theorem 13.22. *There is a parameterized reduction from* Short Turing Machine Acceptance *to* Independent Set.

Proof. Let (M, x, k) be an instance of Short Turing Machine Acceptance. We construct an equivalent instance (G, k') of Independent Set in the following way. By minimal modifications of the Turing machine M, we may assume that whenever it reaches an accepting state, it stays there. Therefore, we may assume that the computation path is infinite and the question is whether M happens to be in an accepting state after step k. Note that during the first k steps, the Turing machine can visit only positions 0, 1, ..., $k+1$ of the tape, hence it is sufficient to "simulate" what happens on these positions. The vertices of G are

- $a_{i,j,\sigma}$ for every $0 \leq i \leq k$, $0 \leq j \leq k+1$, $\sigma \in \Sigma \cup \{\$, \square\}$ and
- $b_{i,j,\delta}$ for every $0 \leq i \leq k$, $0 \leq j \leq k+1$, $\delta \in \Delta$.

The intuitive meaning of $a_{i,j,\sigma}$ being in the independent set is that after step i, the symbol at position j of the tape is σ and the head is *not* on this position. The intuitive meaning of $b_{i,j,\delta}$ being in the independent set is that after step i, the head is at position j, and the next transition is δ; in particular, if, say, $\delta = (q, \sigma, q', \sigma', \leftarrow) \in \Delta$, then this means that after step i, the symbol at position j of the tape is σ and the internal state is q. Let $A_{i,j} = \{a_{i,j,\sigma} : \sigma \in \Sigma \cup \{\$, \square\}\}$, let $B_{i,j,q,\sigma}$ contain every $b_{i,j,\delta}$ where $\delta \in \Delta$, the first coordinate of δ is q, and the second coordinate of δ is σ. Let $B_{i,j,q,\_} = \bigcup_{\sigma \in \Sigma \cup \{\$, \square\}} B_{i,j,q,\sigma}$, let $B_{i,j,\_,\sigma} = \bigcup_{q \in Q} B_{i,j,q,\sigma}$, and let $B_{i,j} = \bigcup_{q \in Q} \bigcup_{\sigma \in \Sigma \cup \{\$, \square\}} B_{i,j,q,\sigma,}$. Let $k' = (k+1)(k+2)$.

We remove some of the vertices of the graph to enforce certain boundary conditions:

(i) Let us remove $A_{0,1} \cup (B_{0,1} \setminus B_{0,1,s_0,\sigma})$, where σ is the first symbol under the head in the initial position. This ensures that a vertex $b_{0,1,\delta} \in B_{0,1,s_0,\sigma}$ of the independent set correctly describes the initial state of the Turing machine.

(ii) For every $2 \leq j \leq k+1$, let us remove $(A_{0,j} \setminus \{a_{0,j,\sigma}\}) \cup B_{0,j}$, where σ is the j-th symbol of the tape in the initial configuration. This ensures that the initial state of the input tape is correctly represented by the independent set.
(iii) For every $0 \leq i \leq k$, let us remove $(A_{i,0} \setminus \{a_{i,0,\$}\}) \cup (B_{i,0} \setminus B_{i,0,\_,\$})$. This ensures that the symbol at position 0 of the tape is always \$.
(iv) For every $0 \leq j \leq k+1$, let us remove $B_{k,j,q,\_}$ for every $q \notin F$. This ensures that the vertex $b_{k,j,\delta}$ representing the state of the Turing machine after step k represents an accepting state.

To enforce the interpretation of the vertices described above, we add edges the following way.

(i) For every $0 \leq i \leq k$, $0 \leq j \leq k+1$, we make $A_{i,j} \cup B_{i,j}$ a clique. This ensures that the only way to have an independent set of size $k' = (k+1)(k+2)$ is to select exactly one vertex from each $A_{i,j} \cup B_{i,j}$.
(ii) For every $0 \leq i \leq k$, we make $\bigcup_{j=0}^{k+1} B_{i,j}$ a clique. This ensures that, for every i, at most one vertex of the independent set can be from some $B_{i,j}$, the rest has to be from the $A_{i,j}$'s, corresponding to the fact that the head of the Turing machine is at one position after step i.
(iii) For every $0 \leq i < k$, $0 \leq j \leq k+1$, and $\sigma \in \Sigma \cup \{\$, \square\}$, we make $a_{i,j,\sigma}$ adjacent to every vertex of $(A_{i+1,j} \setminus \{a_{i+1,j,\sigma}\}) \cup (B_{i+1,j} \setminus B_{i+1,j,\_,\sigma})$. This ensures that if the head of the Turing machine is *not* at position j after step i, then the same symbol σ appears at position j also after step $i+1$. That is, the vertex selected from $A_{i+1,j} \cup B_{i+1,j}$ should be either $a_{i+1,j,\sigma}$ or a vertex from $B_{i+1,j,\_,\sigma}$.
(iv) For every $0 \leq i < k$, $0 \leq j \leq k+1$, and $\delta \in \Delta$, we add edges to $b_{i,j,\delta}$ in the following way:

- if $\delta = (q, \sigma, q', \sigma', \leftarrow)$, then $b_{i,j,\delta}$ is adjacent to every vertex of $A_{i+1,j-1} \cup (B_{i+1,j-1} \setminus B_{i+1,j-1,q',\_}) \cup (A_{i+1,j} \setminus \{a_{i+1,j,\sigma'}\})$ (note that such a left move is not possible for $j = 0$, since then $b_{i,j,\delta} \in B_{i,j,\_,\$}$ and no left move is defined on symbol \$. This ensures that if $b_{i,j,\delta}$ is in the independent set, then the independent set represents that after step $i+1$, the head is at position $j-1$, the state is q', and the symbol at position j is σ'.
- if $\delta = (q, \sigma, q', \sigma', -)$, then $b_{i,j,\delta}$ is adjacent to every vertex of $A_{i+1,j} \cup (B_{i+1,j} \setminus B_{i+1,j,q',\sigma'})$,
- if $\delta = (q, \sigma, q', \sigma', \rightarrow)$, then $b_{i,j,\delta}$ is adjacent to every vertex of $(A_{i+1,j} \setminus \{a_{i+1,j,\sigma'}\}) \cup A_{i+1,j+1} \cup (B_{i+1,j+1} \setminus B_{i+1,j+1,q',\_})$.

As every $b_{i,j,\delta}$ is fully connected to one of $A_{i+1,j-1}$, $A_{i+1,j}$, or $A_{i+1,j+1}$, these edges ensure that if some $b_{i,j,\delta}$ is in the independent set, then a vertex of $B_{i+1,j-1} \cup B_{i+1,j} \cup B_{i+1,j+1}$ is also in the independent set, that is, a position of the head and a state is defined after step $i+1$ as well. Moreover, this new state and position is the result of a valid transition $\delta \in \Delta$.

We claim that M can be in an accepting state on input x after k steps if and only if G has an independent set of size k'. For the first direction, if M has an accepting path, then we can select k' vertices according to the interpretation of the vertices $a_{i,j}$, $b_{i,j,\delta}$ described above. Then it can be verified that this set remains independent after adding the edges introduced in (i)-(iv).

For the other direction, suppose that G has an independent set I of size k'. This is only possible if it contains exactly one vertex from $A_{i,j} \cup B_{i,j}$ for every $0 \leq i \leq k$, $0 \leq j \leq k+1$. We claim that, for every $0 \leq i \leq k$, the set I_i of $k+2$ vertices selected from $A_{i,0} \cup B_{i,0}, \ldots, A_{i,k+1} \cup B_{i,k+1}$ describe a configuration C_i of the Turing machine, and these $k+1$ configurations together describe an accepting computation path. The way we removed vertices ensures the statement for $i = 0$, and in fact the configuration C_0 corresponding to I_0 is the initial state of the Turing machine and the tape. Let us prove the statement by induction on i: suppose that the configurations $C_0, \ldots, C_i$ describe the first i steps of a computation path. Then the way we have added edges ensure that I_{i+1} describe a configuration C_{i+1} that can follow C_i. In particular, as $B_{i+1,0} \cup \cdots \cup B_{i+1,k+1}$ is a clique, at most one vertex of I_{i+1} can describe the position of the head. One vertex of I_i describes the position of the head after step i, that is, I_i contains a vertex of $B_{i,j}$ for some $0 \leq j \leq k+1$. Then the edges introduced in (iv) ensure that I_{i+1} also contains at least one vertex describing the position of the head: every vertex of $B_{i,j}$ is fully connected to either $A_{i+1,j-1}$, $A_{i+1,j}$, or $A_{i+1,j+1}$, which forces the selection of a vertex from $B_{i+1,j-1}$, $B_{i+1,j}$, or $B_{i+1,j+1}$, respectively. Therefore, exactly one vertex of I_{i+1} describes the position of the head. This means that we can define the configuration C_{i+1} based on I_{i+1}, and it can be verified (by analysing the edges added in (iii)-(iv)) that it is indeed a configuration that can follow C_i. □

It is not difficult to give a reduction in the reverse direction: from Independent Set to Short Turing Machine Acceptance (Exercise 13.8), showing the W[1]-completeness of the latter problem.

Theorem 13.23. Short Turing Machine Acceptance *is* W[1]*-complete.*

*13.5 Problems complete for W[1] and W[2]

In Section 13.2, we have seen a set of problems that are at least as hard as Clique or Dominating Set, and therefore W[1]-hard or W[2]-hard, respectively. Now we complete the picture by showing that those problems are actually W[1]-complete or W[2]-complete.

Clique and Independent Set are both W[1]-complete (Theorem 13.18). We have reduced Independent Set to Partial Vertex Cover in Theorem 13.6, showing that Partial Vertex Cover is W[1]-hard. To show

membership in W[1], we can reduce Partial Vertex Cover to Short Turing Machine Acceptance.

Theorem 13.24. *There is a parameterized reduction from* Partial Vertex Cover *to* Short Turing Machine Acceptance.

Proof (sketch). The idea is the following. We encode the input instance to Partial Vertex Cover using the states and transitions of the output Turing machine. Moreover, we take a large alphabet that includes a distinct symbol for each vertex of the input graph. In the first k steps of the Turing machine, a set S of k vertices is guessed and written on the tape. The remaining steps are deterministic. The Turing machine checks whether the k vertices are distinct and counts the total degree of the k vertices. The total degree can be more than the number of edges covered by S, as edges with both endpoints in S are counted twice. Therefore, the Turing machine counts the number of edges induced by S: it reads every pair of vertices from the tape, keeps it in the internal state and increases the counter if they are adjacent. In particular, the input graph is encoded by the transition function: for every state representing a pair of vertices and the value of the counter, the transition function defines a new state where the counter is increased by one (if the two vertices are adjacent) or has the same value (if the two vertices are nonadjacent). Finally, the Turing machine accepts if the total degree minus the number of induced edges is at least s. It is not difficult to implement this procedure with a Turing machine that takes $f(k)$ steps, the details are left to the reader (Exercise 13.10). □

An instance $(G, k, (V_1, \ldots, V_k))$ of Multicolored Clique can be easily reduced to Clique: all we need to do is to remove those edges whose both endpoints are in the same V_i. Then any k-clique has exactly one vertex from each V_i. Therefore, Multicolored Clique and Multicolored Independent Set are both W[1]-complete.

Theorem 13.25. *The following parameterized problems are* W[1]*-complete:*

- Clique
- Multicolored Clique
- Independent Set
- Multicolored Independent Set
- Partial Vertex Cover
- Short Turing Machine Acceptance

We have seen in Theorem 13.21 that Dominating Set, Set Cover, and Hitting Set are W[2]-complete. Theorem 13.15 presented a reduction from Dominating Set to Connected Dominating Set, hence Connected Dominating Set is W[2]-hard. Let us try to show a reduction in the reverse direction to prove that Connected Dominating Set is W[2]-complete. For this purpose, we introduce an auxiliary problem first. The Dominating Set with Pattern is a version of Dominating Set where we prescribe what

subgraph the dominating set should induce. Formally, the input is a graph G, an integer k, and a graph H on the vertex set $[k]$. The task is to find a k-tuple $(v_1, \ldots, v_k)$ of distinct vertices such that (1) they form a dominating set and (2) vertices v_i and v_j are adjacent if and only if ij is an edge of H. It is easy to observe that Dominating Set with Pattern is W[2]-hard: the reduction of Theorem 13.15 from Dominating Set to Connected Dominating Set has the property that if the created instance has a connected dominating set of size k, then it has a dominating set of size k inducing a clique (Exercise 13.12). We can show that Dominating Set with Pattern is in W[2] by reducing it to Dominating Set.

Theorem 13.26. *There is a parameterized reduction from* Dominating Set with Pattern *to* Dominating Set.

Proof. Let (G, k, H) be an instance of Dominating Set with Pattern. We construct a graph G' as follows.

(i) For every vertex $v \in V(G)$, we introduce k vertices $v_1, \ldots, v_k$.
(ii) For every $1 \leq i \leq k$, the set $V_i = \{v_i \ : \ v \in V(G)\}$ forms a clique.
(iii) For every $1 \leq i \leq k$, we introduce two vertices x_i, y_i, and make them adjacent to every vertex of V_i.
(iv) For every $v \in V(G)$, we introduce a vertex v^* and make it adjacent to every vertex of $\{u_i \ : \ u \in N_G[v], 1 \leq i \leq k\}$.
(v) For every $1 \leq i < j \leq k$ such that ij is an edge of H and for every pair (a, b) of nonadjacent vertices in G (including the possibility $a = b$), we introduce a vertex $w_{i,j,a,b}$ and make it adjacent to every vertex of $(V_i \cup V_j) \setminus \{a_i, b_j\}$.
(vi) For every $1 \leq i < j \leq k$ such that ij is *not* an edge of H and for every pair (a, b) such that either a and b are adjacent in G or $a = b$, we introduce a vertex $w_{i,j,a,b}$ and make it adjacent to every vertex of $(V_i \cup V_j) \setminus \{a_i, b_j\}$.

Let $(v^1, \ldots, v^k)$ be a k-tuple of distinct vertices of G forming a dominating set such that v_i and v_j are adjacent if and only if ij is an edge of H. We claim that $S' = \{v_1^1, \ldots, v_k^k\}$ is a dominating set of G'. Vertex v_i^i dominates every vertex in $V_i \cup \{x_i, y_i\}$. Consider a vertex v^* introduced in (iv). There is a vertex v^i dominating v in G, hence v_i^i dominates v^* in G'. Consider now a vertex $w_{i,j,a,b}$ introduced in (v). As ij is an edge of H, vertices v^i and v^j are adjacent. As either $a = b$ or a and b are not adjacent, assertions $v^i = a$ and $v^j = b$ cannot be both true, and hence assertions $v_i^i = a_i$ and $v_j^j = b_j$ cannot be both true. This implies that $(V_i \cup V_j) \setminus \{a_i, b_j\}$ contains at least one of v_i^i and v_j^j, and we can conclude that $w_{i,j,a,b}$ is dominated by S'. We argue similarly for a vertex $w_{i,j,a,b}$ introduced in (vi): now, as ij is not an edge of H, vertices v^i and v^j are not adjacent. As a and b are adjacent or $a = b$, we have again that $(V_i \cup V_j) \setminus \{a_i, b_j\}$ contains at least one of v_i^i and v_j^j, dominating $w_{i,j,a,b}$.

For the reverse direction, let S' be a dominating set of G'. It has to contain at least one vertex from the set $V_i \cup \{x_i, y_i\}$ to dominate the vertices x_i and y_i. As these sets are disjoint for distinct values of i, we can conclude that S' contains exactly one vertex from $V_i \cup \{x_i, y_i\}$; in fact, exactly one vertex from V_i, as x_i and y_i do not dominate each other. Let $S' = \{v_1^1, \ldots, v_k^k\}$, where $v_i^i \in V_i$. We claim first that $S = \{v^1, \ldots, v^k\}$ is a dominating set of G: if a vertex $v \in V(G)$ is not dominated by S, then vertex v^* is not dominated by S' in G'. Next we claim that if ij is an edge of H, then v^i and v^j are different and adjacent in G. Suppose that ij is an edge of H for some $1 \leq i < j \leq k$. If v^i and v^j are equal or not adjacent, then we have introduced a vertex w_{i,j,v^i,v^j} that is adjacent only to $(V_i \cup V_j) \setminus \{v_i^i, v_j^j\}$, hence it is not dominated by S' in G', a contradiction. Similarly, we claim that if ij is not an edge of H, then v^i and v^j are two distinct nonadjacent vertices. Indeed, if v^i and v^j are adjacent or equal, then we have introduced a vertex w_{i,j,v^i,v^j} that is adjacent only to $(V_i \cup V_j) \setminus \{v_i^i, v_j^j\}$, hence not dominated by S'. In particular, the two claims imply that $(v^1, \ldots, v^k)$ are distinct vertices, as any two of them are either connected by an edge or nonadjacent distinct vertices. Therefore, we get that $(v^1, \ldots, v^k)$ is a k-tuple of distinct vertices forming a dominating set in G and inducing the required pattern. □

In the light of Theorem 13.26, we can show that Connected Dominating Set is in W[2] by reducing it to Dominating Set with Pattern. A solution of Connected Dominating Set induces a connected graph on k vertices and there are $f(k) = 2^{\mathcal{O}(k^2)}$ connected graphs on k labelled vertices. Therefore, it is tempting to say that Connected Dominating Set can be reduced to Dominating Set with Pattern by trying every possible such pattern and applying an algorithm for Dominating Set with Pattern. However, this is not a many-one parameterized reduction, as defined in Definition 13.1: given an instance I of Connected Dominating Set, instead of creating a single equivalent instance I' of Dominating Set with Pattern, we create an set $I'_1, \ldots, I'_t$ of instances of Dominating Set with Pattern such that I is a yes-instance if and only if at least one I'_i is a yes-instance. This is an example of a so-called *parameterized Turing reduction.* Without formalizing the details, a parameterized Turing reduction from A to B is an algorithm that solves an instance of A by having access to an "oracle" that can solve instances of B in constant time. The reduction has the property that, given an instance (x, k) of A, the running time is $f(k)|x|^{\mathcal{O}(1)}$ and the parameter of every created instance of B can be bounded by $f(k)$.

Turing reductions transfer fixed-parameter tractability the same way as many-one reductions: one could state an analogue of Theorem 13.2 saying that parameterized Turing reduction from A to B implies that if B is FPT, then A is FPT as well. Therefore, from the point of view of giving negative evidence for fixed-parameter tractability, Turing reductions are just as good as many-one reductions. However, we defined the W-hierarchy by closure under many-one reductions, thus to complete the picture, we need a many-one re-

duction to prove the membership of Connected Dominating Set in W[2]. It is possible to modify the reduction of Theorem 13.26 to output a single instance of Dominating Set for a collection of patterns (Exercise 13.13). Alternatively, we can use the definition of W[2] directly and reduce Connected Dominating Set to circuit satisfiability problem.

Theorem 13.27. Connected Dominating Set *is in* W[2].

Proof. Given an instance (G, k) of Connected Dominating Set, we construct an equivalent instance (C, k) of Weighted Circuit Satisfiability. Let $v_1, \ldots, v_n$ be the vertices of G. The input nodes of the constructed circuit are $x_{i,j}$ for $1 \leq i \leq k$, $1 \leq j \leq n$. The interpretation of $x_{i,j} = 1$ is that the i-th vertex of the solution is v_j.

(i) For $1 \leq i \leq k$, node z_i is the disjunction of $\{x_{i,j} : 1 \leq j \leq n\}$ and O_1 is the conjunction of all the z_i-s. Then $O_1 = 1$ expresses the requirement that there are integers $s_1, \ldots, s_k$ such that $x_{1,s_1}, \ldots, x_{k,s_k}$ are exactly the input nodes with value 1, describing a tuple $(v_{s_1}, \ldots, v_{s_k})$ of k vertices.

(ii) For $1 \leq j \leq i$, node w_j expresses that vertex v_j is dominated in the solution: it is the disjunction of the input nodes $\{x_{i,j'} : 1 \leq i \leq k, v_{j'} \in N[v_j]\}$.

(iii) Node O_2 expresses that the input represents a dominating set: it is the conjunction of w_j for every $1 \leq j \leq n$.

(iv) For every P such that $\emptyset \subsetneq P \subsetneq [k]$, node c_P expresses that there is an edge between $\{v_{s_i} : i \in P\}$ and $\{v_{s_i} : i \notin P\}$. For every $i_1 \in P$, $i_2 \in [k] \setminus P$, and pair (v_{j_1}, v_{j_2}) of adjacent or equal vertices in G, we introduce a node e_{P,i_1,i_2,j_1,j_2} that is the conjunction of x_{i_1,j_1} and x_{i_2,j_2}; the node c_P is the disjunction of all these nodes.

(v) Node O_3 expresses that $(v_{s_1}, \ldots, v_{s_k})$ induces a connected graph: it is the conjunction of all the nodes c_P.

(vi) The output node is the conjunction of O_1, O_2, and O_3 (more precisely, there is a small node O' that the conjunction of O_1 and O_2, and the output node is a small node that is the conjunction of O' and O_3).

Observe that the constructed circuit C has constant depth (independent of k) and weft 2: a path from an input to the output contains either

- a large node z_i and the large node O_1,
- or a large node w_j and the large node O_2,
- or a large node c_P and the large node O_3.

It is easy to see that if $(v_{s_1}, \ldots, v_{s_k})$ is a connected dominating set, then setting $x_{1,s_1}, \ldots, x_{k,s_k}$ to 1 satisfies the circuit. In particular, as $(v_{s_1}, \ldots, v_{s_k})$ induces a connected graph, for every $\emptyset \subsetneq P \subsetneq [k]$ there has to be an $i_1 \in P$ and $i_2 \in [k] \setminus P$ such that $v_{s_{i_1}}$ and $v_{s_{i_2}}$ are adjacent. This means that the node $e_{i_1,i_2,s_{i_1},s_{i_2}}$ has value 1, implying that c_P is 1 as well.

The reverse direction is also easy to show: it is clear that any satisfying assignment describes a k-tuple $(v_{s_1}, \ldots, v_{s_k})$ of vertices (possibly with repeated vertices) that forms a dominating set of the graph. If these vertices did not induce a connected graph, then they can be partitioned into two disconnected parts, that is, there is a $\emptyset \subset P \subset [k]$ such that there is no edge between v_{i_1} and v_{i_2} for any $i_1 \in P$ and $i_2 \in [k] \setminus P$. Then c_P has value 0, implying that O_3 has value 0, and hence the output has value 0 as well. □

We have shown in Section 13.2 that Dominating Set on undirected graphs, general directed graphs, tournaments, and Set Cover (or Hitting Set) are equally hard (see also Exercises 13.2). We summarize these results in the following theorem.

Theorem 13.28. *The following problems are* W[2]*-complete:*

- Dominating Set
- Dominating Set *for directed graphs*
- Dominating Set on Tournaments
- Connected Dominating Set
- Set Cover
- Hitting Set
- Weighted 2-normalized Satisfiability
- Weighted Monotone 2-normalized Satisfiability
- Weighted Monotone 3-normalized Satisfiability

13.6 Parameterized reductions: further examples

In Section 13.2 we have presented reductions showing that basic problems such as Dominating Set are unlikely to be fixed-parameter tractable. The goal of this section is to explain, via further examples, the different types of parameterized reductions one can encounter in the parameterized complexity literature.

13.6.1 Reductions keeping the structure of the graph

The simplest type of reduction is when there is relatively simple correspondence between the input graph and the output graph. This is the case, for example, in the trivial reduction from Clique to Independent Set, or when reducing Dominating Set to Connected Dominating Set (Theorem 13.15). In other examples, the reduction may involve more complicated operations than just taking the complement of the graph (e.g., subdividing

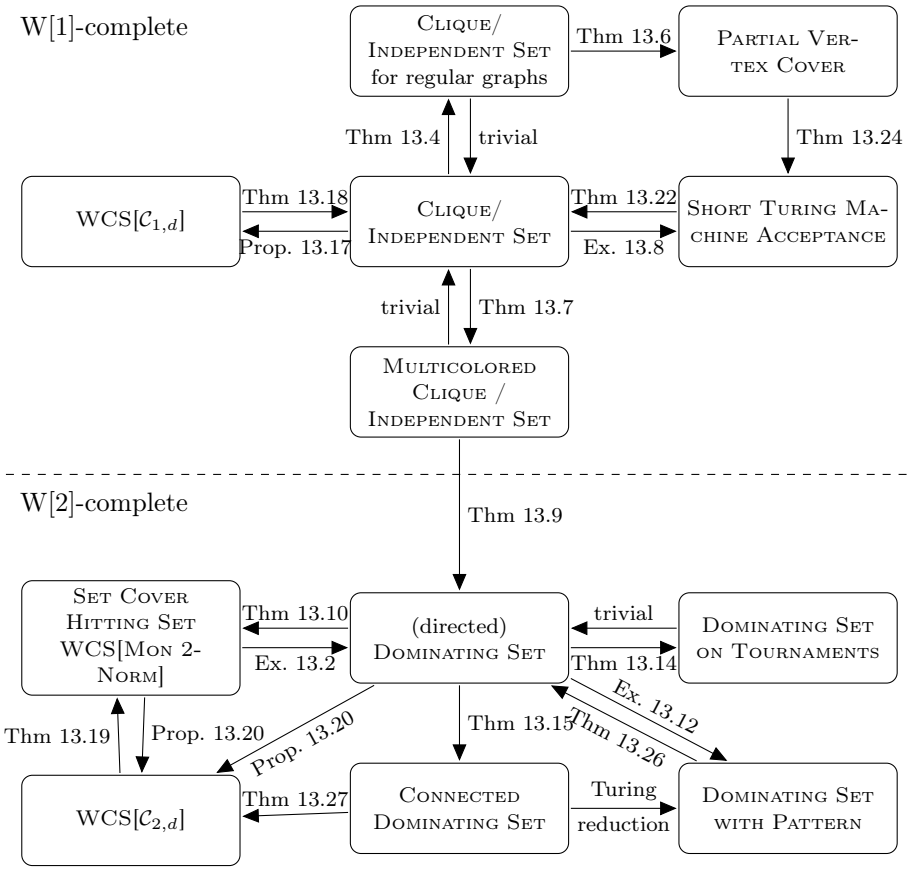


Fig. 13.4: W[1]-complete and W[2]-complete problems in Sections 13.2–*13.5 and the reductions between them. WCS[Mon 2-Norm] is short for Weighted Monotone 2-normalized Satisfiability.

edges, replacing vertices/edges with certain fixed subgraphs etc.), but the correspondence between the vertices/edges of the two graphs is fairly straightforward. The following reduction is an example where the correspondence is still relatively clear, although not completely trivial. In the Balanced Vertex Separator problem, we are given a graph G and integer k; the task is to find a set S of at most k vertices such that every component of $G - S$ has at most $|V(G)|/2$ vertices.

Theorem 13.29. Balanced Vertex Separator *is* W[1]*-hard.*

Proof. We give a parameterized reduction from Clique. Let (G, k) be an instance of Clique. Let $n = |V(G)|$ and $m = |E(G)|$. We may assume

that n is sufficiently large compared to k, for example, $n \geq 4k^2$: adding $4k^2$ isolated vertices to the graph G does not change the problem. Let us set $\ell = n + m - 2(k + \binom{k}{2}) \geq 0$. We construct a graph G' as follows.

(i) The vertex set of G' contains $V(G)$, which forms a clique in G'.
(ii) For every edge $e = uv$ of G, we introduce a vertex w_e in G' and make it adjacent to u and v.
(iii) We introduce a clique K on ℓ new vertices as another component of G'.

The output instance is (G', k).

The intuition behind the construction is that if we remove a set S of vertices from the clique formed by $V(G)$ in G', then it creates some number of components of size 1, corresponding to the edges induced by S in G. Therefore, if we want to maximize the number of these components appearing (to reduce the size of the other components), then we need a set S that maximizes the number of edges induced in G, that is, forms a k-clique.

Note that G' has $n+m+\ell = 2(n+m-k-\binom{k}{2}) \geq 2\ell$ vertices. Suppose that S is a k-clique of G. Then $G' \setminus S$ has the following components: $\binom{k}{2}$ components of size 1, the clique K of size $\ell \leq |V(G')|/2$, and one component containing the remaining $(n + m + \ell) - |S| - \binom{k}{2} - \ell = n + m - k - \binom{k}{2} = |V(G')|/2$ vertices, that is, every component of $G' - S$ has size at most $|V(G')|/2$.

For the reverse direction, suppose that S is a set of at most k vertices in G' such that every component of $G' - S$ has size at most $|V(G')|/2 = n + m - k - \binom{k}{2}$. First we show that it can be assumed that $S \subseteq V(G)$. Clearly, it makes no sense to remove any vertex of the clique K of size ℓ, as the size of this component is already at most $|V(G')|/2$. Suppose that $w_e \in S$ for some edge $e = uv$. If $u, v \in S$, then we many omit w_e from S: now w_e becomes an isolated vertex in $G - (S \setminus \{w_e\})$. If, say, $u \notin S$, then $(S \setminus \{w_e\}) \cup \{u\}$ is also a solution: omitting w_e from S increases the size of the component of u by 1 (note that u and v are adjacent, hence they cannot be in two different components of $G' - S$), and introducing u into S decreases the size of this component by 1. Therefore, we may assume $S \subseteq V(G)$.

Consider the component C of $G' - S$ that contains the clique $V(G) \setminus S$ (as we have assumed that $n = |V(G)| > k$, the component C is nonempty). If an edge $e \in E(G)$ has an endpoint not in S, then vertex $w_e \notin S$ of G' is also in the component C. Thus if we denote by p the number of edges induced by S in G, then component C has size $(n - |S|) + m - p$. Then the bound on the size of C implies $(n - |S|) + m - p \leq |V(G')|/2 = n + m - k - \binom{k}{2}$, that is, we get $|S| + p \geq k + \binom{k}{2}$. As $|S| \leq k$, this is only possible if $|S| = k$ and S induces a clique in G. □

13.6.2 Reductions with vertex representation

The most common technique for NP-hardness proofs is to represent parts of the original instance by gadgets. When reducing a graph problem to another graph problem, this is done by replacing each vertex and each edge of the original graph by a "gadget": a small graph satisfying certain properties. Then it is argued that these gadgets have properties ensuring that the solutions of the constructed instance correspond to the solutions of the original instance. A typical example of a reduction of this type is reducing Vertex Cover to Hamiltonian Cycle. Similarly, when reducing from 3-SAT, one may use gadgets representing the variables and clauses of the original formula. A typical example is reducing 3-SAT to Clique.

Representing vertices and edges of the original graph (or variables and clauses of the original formula) by gadgets is usually not a suitable technique for parameterized reductions. The problem is that, in most cases, by introducing a new gadget, one has to increase the parameter of the constructed instance. Therefore, the parameter of the new instance would be proportional to the number of vertices/edges/variables/clauses of the original instance, which can be much larger than the parameter of the original instance, violating the second requirement of Definition 13.1.

A typical parameterized reduction from Clique uses gadgets in a very different way than NP-hardness proofs. Instead of creating a gadget for each vertex/edge of the input graph, we create k gadgets, one for each vertex of the *solution.* If we can ensure that the new parameter is proportional to the number of gadgets, then we satisfy the requirement that the new parameter is bounded by a function of the original parameter.

A consequence of using only k gadgets is that we need very different, much more powerful gadgets than in typical NP-hardness proofs. For example, in a polynomial-time reduction from, say, Vertex Cover, the gadget needs to be able to represent only a constant number of different states: each vertex is either selected or not, and each edge can be covered in three possible ways (first endpoint, second endpoint, both endpoints). On the other hand, if we want a gadget to represent one of the vertices of the solution, then we need a gadget that is able to represent n different states, where n is the number of vertices of the original graph.

Theorem 13.9, the reduction from Multicolored Independent Set to Dominating Set, can be considered as an example of representing the vertices of the solution by gadgets. Selecting one vertex of the clique V_i of G' corresponds to selecting a vertex of V_i into the independent set. Thus we may view the clique V_i as a gadget with $|V_i|$ possible states. Then the

vertices w_e "verify" that the states of the gadgets $V_1, \ldots, V_k$ correspond to an independent set of G.

The following reduction demonstrates the same principle: we want to represent the k vertices of the solution by k gadgets and, for any two of these gadgets, we want to ensure that they represent nonadjacent vertices. This reduction also serves as an example for a reduction to a problem where the parameter is not related to the solution size, but it is a structural parameter measuring some property of the instance (in our case, the treewidth). In such a reduction, what we need to ensure is that the structural parameter is proportional to the number of gadgets.

List Coloring is a generalization of Vertex Coloring: given a graph G, a set of colors C, and a list function $L : V(G) \to 2^C$ (that is, a subset of colors $L(v)$ for each vertex v), the task is to assign a color $c(v) \in L(v)$ to each vertex $v \in V(G)$ such that adjacent vertices receive different colors. Vertex Coloring on a graph with treewidth w can be solved in time $2^{\mathcal{O}(w \log w)} \cdot n^{\mathcal{O}(1)}$ by standard dynamic programming techniques (Theorem 7.10). However, the straightforward application of dynamic programming yields only a $n^{\mathcal{O}(w)}$ time algorithm for the more general List Coloring problem (Exercise 7.20). The following theorem shows that there is a reason for that: the problem is W[1]-hard parameterized by treewidth.

Theorem 13.30. List Coloring *is* W[1]*-hard when parameterized by treewidth.*

Proof. We present a parameterized reduction from Multicolored Independent Set. Let $(G, k, (V_1, \ldots, V_k))$ be an instance of Multicolored Independent Set. The set C of colors corresponds to $V(G)$. We construct a graph G' in the following way (see Fig. 13.5).

(i) For every $1 \leq i \leq k$, we introduce a vertex u_i with $L(u_i) = V_i$.
(ii) For every $1 \leq i < j \leq k$ and for every edge ab with $a \in V_i$, $b \in V_j$, we introduce a vertex $w_{i,j,a,b}$ with $L(w_{i,j,a,b}) = \{a, b\}$ and make it adjacent to u_i and u_j.

Observe that $\{u_1, \ldots, u_k\}$ is a vertex cover of G', hence G' has treewidth at most k. Therefore, the reduction satisfies the requirement that the parameter of the constructed instance is bounded by a function of the parameter of the original instance. Suppose that G has an independent set of size k, consisting of vertices $v_1 \in V_1, \ldots, v_k \in V_k$. We can construct the following list coloring of G'. First, we set the color of u_i to $v_i \in V_i$. Then for every vertex $w_{i,j,a,b}$, it is true that either u_i has a color different from a, or u_j has a color different from b (as a and b are adjacent, while v_i and v_j are not). Therefore, one of the two colors appearing in the list of $w_{i,j,a,b}$ is not used on its neighbors, hence we can extend the coloring to $w_{i,j,a,b}$.

For the reverse direction, suppose that c: $V(G') \to V(G)$ is a list coloring of G'. We claim that $c(u_1) \in L(u_1) = V_1, \ldots, c(u_k) \in L(u_k) = V_k$ is an independent set in G. For a contradiction, suppose that $a = c(u_i)$ and $b = c(u_j)$ are adjacent for some $1 \leq i < j \leq k$. Then the vertex $w_{i,j,a,b}$ exists in

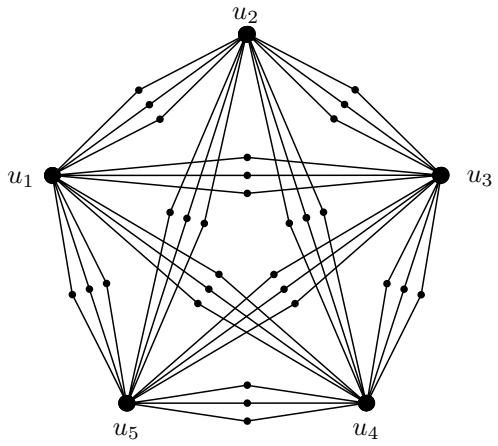


Fig. 13.5: The graph G' of the List Coloring instance constructed in the reduction of Theorem 13.30 for $k = 5$.

G with list $\{a, b\}$. However, this vertex is adjacent to both u_i and u_j, hence $c(w_{i,j,a,b})$ can be neither a nor b, a contradiction. □

13.6.3 Reductions with vertex and edge representation

In reductions from Clique/Independent Set based on vertex representation, we need to enforce that gadgets represent adjacent/nonadjacent vertices. In Theorem 13.9 we enforced this condition by introducing the vertices w_e. Introducing these vertices had no effect on the parameter, as these vertices were supposed to be dominated by the k vertices selected from $V_1, \ldots, V_k$. Similarly, in Theorem 13.30, we introduced the vertices $w_{i,j,a,b}$ to enforce that the coloring on $w_1, \ldots, w_k$ represents an independent set. The introduction of these vertices did not increase the treewidth of the constructed instance. However, there are reductions where it seems difficult to enforce the adjacency/nonadjacency condition on two gadgets. We present two examples of such problems and show how to prove W[1]-hardness in this case by representing both the edges and the vertices of the clique by gadgets.

Example 1: Odd Set **and variants**

In the Odd Set problem, where given a set system $\mathcal{F}$ over a universe U and an integer k, the task is to find a subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has odd size. A natural idea for reducing

an instance $(G, k, (V_1, \ldots, V_k))$ of Multicolored Independent Set to an instance of Odd Set would be to let U be $V(G)$, keep the same parameter k, and introduce the sets $V_1, \ldots, V_k$. The only way a set S of size at most k can have odd intersection with these k disjoint sets is to have exactly one vertex from each of $V_1, \ldots, V_k$. Now we would like to introduce sets into $\mathcal{F}$ to enforce that the vertex of the solution in V_i is not adjacent to the vertex of the solution in V_j. If $a \in V_i$ and $b \in V_j$ are adjacent, then introducing the set $\{a, b\}$ into $\mathcal{F}$ certainly prevents the possibility that both a and b is in a solution S. However, it also prevents the possibility that neither of a and b is in S, which may very well be the case for a solution. The reader may try other simply tricks, but it seems that the parity requirement is not strong enough to represent the complicated condition that two vertices are not adjacent.

We get around this difficulty by a reduction from Multicolored Clique where we represent both the vertices and the edges of the clique, that is, we have k vertex gadgets and $\binom{k}{2}$ edge gadgets. Then what we need to ensure is that if an edge gadget represents the edge uv, then the corresponding vertex gadgets represent u and v, respectively. As we shall see, such conditions are much easier to express and enforce.

Theorem 13.31. Odd Set *is* W[1]*-hard.*

Proof. We present a parameterized reduction from Multicolored Clique. Let $(G, k, (V_1, \ldots, V_k))$ be an instance of Multicolored Clique. Let $E_{i,j}$ be the set of edges between V_i and V_j, and let $E_{i,j,v}$ be the subset of $E_{i,j}$ consisting of the edges incident to v. We construct an instance $(\mathcal{F}, U, k')$ of Odd Set as follows.

(i) The universe is $U := \bigcup_{i=1}^{k} V_i \cup \bigcup_{1 \leq i < j \leq k} E_{i,j}$.
(ii) Let $k' := k + \binom{k}{2}$.
(iii) For every $1 \leq i \leq k$, let $\mathcal{F}$ contain V_i, and for every $1 \leq i < j \leq k$, let $\mathcal{F}$ contain $E_{i,j}$.
(iv) For every $1 \leq i < j \leq k$, let $\mathcal{F}$ contain $E'_{i,j,v} := E_{i,j,v} \cup (V_i \setminus \{v\})$ for every $v \in V_i$ and $E'_{i,j,w} := E_{i,j,w} \cup (V_j \setminus \{w\})$ for every $w \in V_j$.

The intuitive idea of the reduction is the following. The solution needs to select one vertex from each of $V_1, \ldots, V_k$ (describing k vertices of G) and one vertex from each $E_{i,j}$ (describing $\binom{k}{2}$ edges of G). Then the sets introduced in (iv) ensure that the vertices selected from V_i and V_j are the endpoints of the edge represented by the vertex selected from $E_{i,j}$. More precisely, the set $E'_{i,j,v}$ ensures that exactly one of $V_i \setminus \{v\}$ and $E_{i,j,v}$ contains a vertex of the solution, that is, $v \in V_i$ is in the solution if and only if the solution contains a vertex of $E_{i,j,v}$.

Formally, let $v_1 \in V_1, \ldots, v_k \in V_k$ be a k-clique in G and let $e_{i,j}$ be the edge between v_i and v_j. We claim that the set $S := \bigcup_{i=1}^{k} \{v_i\} \cup \bigcup_{1 \leq i < j \leq k} \{e_{i,j}\}$

has odd intersection with every set in $\mathcal{F}$. This is clearly true for the sets V_i and $E_{i,j}$. Consider now a set $E'_{i,j,v}$ introduced in (iv). Suppose that $v \in V_i$ (the argument is similar for the case $v \in V_j$). If the endpoint of $e_{i,j}$ in V_i is v, then $v = v_i$ and $e_{i,j} \in E_{i,j,v}$ hold and hence $S \cap E'_{i,j,v} = \{e_{i,j}\}$, which has odd size. If the endpoint of $e_{i,j}$ in V_i is not v, then $v \neq v_i$ and $e_{i,j} \notin E_{i,j,v}$, hence $S \cap E'_{i,j,v} = \{v_i\}$, which is again odd.

For the reverse direction, let $S \subseteq U$ be a set of size at most k'. As the $k' = k + \binom{k}{2}$ sets introduced in (iii) are disjoint, each of them contains exactly one vertex of S. Let $S \cap V_i = \{v_i\}$ and $S \cap E_{i,j} = \{e_{i,j}\}$. We claim that $v_1 \in V_1$, $\ldots$, $v_k \in V_k$ is a k-clique in G and $e_{i,j}$ is the edge between v_i and v_j. Suppose that v_i is not an endpoint of $e_{i,j}$. Then $v_i \notin E'_{i,j,v_i}$ and $e_{i,j} \notin E'_{i,j,v_i}$ hold and hence $S \cap E'_{i,j,v_i} = \emptyset$, a contradiction. The argument is similar if v_j is not an endpoint of $e_{i,j}$. Therefore, $e_{i,j}$ is the edge between v_i and v_j, hence $\{v_1, \ldots, v_k\}$ indeed induces a clique in G. □

In general, the requirement that $a \in V_i$ and $b \in V_j$ are adjacent can be an arbitrarily complicated binary relation: the bipartite graph between V_i and V_j can be arbitrarily complicated. The key idea in the proof of Theorem 13.31 is that the requirement that $v \in V_i$ is an endpoint of an edge $e \in E_{i,j}$ is a much simpler binary relation: it is a projection, that is, for every $e \in E_{i,j}$, there is only a single $v \in V_i$ that satisfies the requirement. The general principle that we can learn from this example is that, after setting up the gadgets representing the k vertices and the $\binom{k}{2}$ edges of the clique, all we need to ensure that each state of an edge gadget *forces* a particular state for the the two vertex gadgets representing the endpoints of the edge. That is, we do not need to implement arbitrarily complicated relations between the gadgets, only a very specific simple relation requiring that a state of an edge gadget always implies a particular state on a vertex gadget.

The proof of Theorem 13.31 allows us to prove W[1]-hardness for some variants of Odd Set. First, Exact Odd Set is the variant that asks for a solution of size *exactly* k. Observe that, given a yes-instance of Multicolored Clique, the reduction of Theorem 13.31 produces an instance of Odd Set that has a solution of size exactly k'. Therefore, this is also a correct parameterized reduction from Multicolored Clique to Exact Odd Set, proving the W[1]-hardness of the latter problem.

One can define the Even Set and Exact Even Set problems analogously: now each set has to contain an even number of vertices of the solution. Note, however, that the empty set is always a correct solution for Even Set with this definition. Therefore, in the Even Set problem, we require the solution to be a *nonempty* set of at most k elements such that every set in the instance contains an even number of elements of the solution. It is not difficult to reduce Exact Odd Set to Exact Even Set (Exercise 13.19), showing that Exact Even Set is also W[1]-hard. On the other hand, the fixed-parameterized tractability of Even Set is a notorious open question. It is equivalent to other very interesting and deep questions: it can be also formulated as finding the minimum distance of a binary linear code or the

minimum length cycle in a binary matroid. Therefore, settling this question may require a deeper combinatorial understanding and could be more challenging than the simple reduction of Theorem 13.31.

Unique Hitting Set is the variant of Odd Set where we require that each set contains *exactly one* element of the solution; Exact Unique Hitting Set is defined analogously. Looking at the reduction in Theorem 13.31, we can observe that if the Multicolored Clique instance is a yes-instance, then the constructed instance of Odd Set has the property that every set contains in fact only a single element of the solution. Therefore, the proof of Theorem 13.31 actually provides a reduction from Multicolored Clique to Unique Hitting Set or Exact Unique Hitting Set.

Theorem 13.32. *The following problems are* W[1]*-hard:*

- Odd Set
- Exact Odd Set
- Exact Even Set
- Unique Hitting Set
- Exact Unique Hitting Set

Interestingly, one can prove that Unique Hitting Set and Exact Unique Hitting Set is actually W[1]-complete by a reduction to Short Turing Machine Acceptance (Exercise 13.20), but this is open for the other three problems.

It is natural to define analogues of Odd Set (and its variants) on graphs with the closed neighborhoods of the vertices playing the role of the set system, the same way as Dominating Set can be considered as a graph-theoretic analogue of Hitting Set. For example, the graph version of Odd Set asks for a set of at most k vertices such that $N[v]$ contains an odd number of vertices of the solution for every vertex v. The graph version of Exact Unique Hitting Set is also known under the name Perfect Code. If the vertex set of a graph G is the set of all potential codewords and two vertices are adjacent if the corresponding codewords are "close," then Perfect Code asks for a code containing exactly k words such that every word is close to exactly one word in the code. One can show that the graph versions of all five problems considered in Theorem 13.32 are also W[1]-hard (Exercise 13.21).

Example 2: Steiner problems in directed graphs

Given an undirected graph G, a set $K \subseteq V(G)$ of terminals, and an integer ℓ, the Steiner Tree problem asks for a tree H with at most ℓ edges connecting all the terminals. In Sections 6.1.2 and 10.1.2 we gave algorithms for Steiner Tree running in time $3^{|K|}n^{\mathcal{O}(1)}$ and $2^{|K|}n^{\mathcal{O}(1)}$, respectively. One can define similar problems on directed graphs. Given a set of terminals $K \subseteq V(G)$ and a root $r \in V(G)$, Directed Steiner Tree asks for a directed tree rooted at r such that every terminal in K is reachable from r on the tree. This

problem generalizes Steiner Tree to directed graphs in the sense that we are looking for a subgraph that provides reachability from one distinguished vertex to every other terminal. See Exercise 6.7 for a $3^{|K|}n^{\mathcal{O}(1)}$ time algorithm for Directed Steiner Tree. Another, equally natural, generalization is the Strongly Connected Steiner Subgraph problem, where the input is a directed graph G, a set $K \subseteq V(G)$ of terminals, and an integer ℓ; the task is to find a strongly-connected subgraph of G with at most ℓ vertices that contains every vertex of K. This problem generalizes Steiner Tree in the sense that we have to provide a path from any terminal to every other terminal. Note that one can define Strongly Connected Steiner Subgraph by requiring either at most ℓ edges or at most ℓ vertices in the solution. The two versions are different, but here we discuss only the vertex version for simplicity. The parameter we consider is the number ℓ of vertices in the solution. Unlike Directed Steiner Tree, this problem is W[1]-hard and the proof is another example of the edge representation strategy.

Theorem 13.33. Strongly Connected Steiner Subgraph *is* W[1]*-hard.*

Proof. We present a parameterized reduction from Multicolored Clique. Let $(G, k, (V_1, \ldots, V_k))$ be an instance of of Multicolored Clique. Let $E_{i,j}$ be the set of edges between V_i and V_j. We construct an instance (G', K, ℓ) of Strongly Connected Steiner Subgraph as follows (see Fig. 13.6).

(i) G' contains all the vertices of G, a vertex x, vertices $y_{i,j}$ $(1 \leq i < j \leq k)$, and a vertex w_e for every $e \in E(G)$. We define $E'_{i,j} = \{w_e \ : \ e \in E_{i,j}\}$ for $1 \leq i < j \leq k$.
(ii) Let $K := \{x\} \cup \{y_{i,j} \ : \ 1 \leq i < j \leq k\}$ and let $\ell := k + 1 + 2\binom{k}{2} = |K| + k + \binom{k}{2}$.
(iii) For every $1 \leq i \leq k$ and $v \in V_i$, we introduce the edges (x, v) and (v, x).
(iv) For every $1 \leq i < j \leq k$ and $e \in E_{i,j}$, we introduce the edges $(y_{i,j}, w_e)$ and $(w_e, y_{i,j})$.
(v) For every $1 \leq i < j \leq k$ and $e \in E_{i,j}$, if $a \in V_i$ and $b \in V_j$ are the endpoints of e, then we introduce the edges (a, w_e) and (w_e, b).

The intuitive idea of the reduction is the following. The solution can afford to select only one vertex from each of $V_1, \ldots, V_k$ ("vertex gadgets" describing k vertices of G) and one vertex from each $E'_{i,j}$ ("edge gadgets" describing $\binom{k}{2}$ edges of G). Each vertex w_e in $E_{i,j}$ has only one inneighbor and one outneighbor different from $y_{i,j}$, thus if w_e is part of the solution, then those two vertices have to be selected as well. Therefore, the state of each edge gadget forces a state on two vertex gadgets, implying that the k-vertex gadgets describe a k-clique in G.

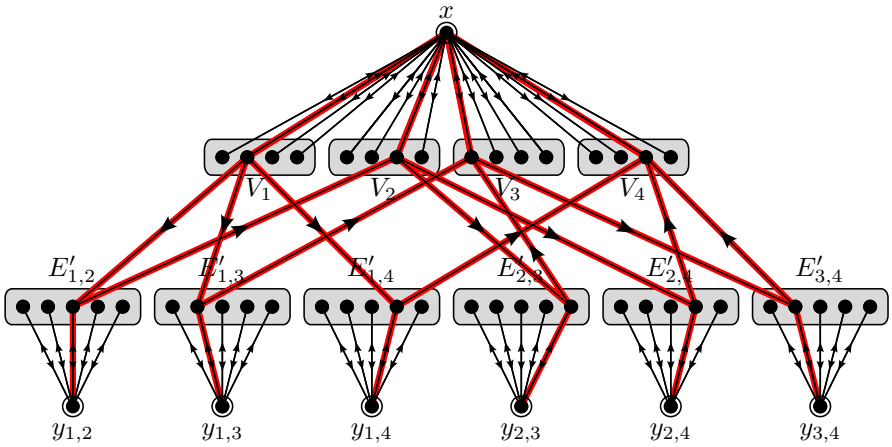


Fig. 13.6: An overview of the reduction in the proof of Theorem 13.33. The circled vertices are the terminals and the red edges show a possible solution. For clarity, we show only those edges of the graph between the V_i's and $E'_{i,j}$ that are in the solution.

Formally, let $v_1 \in V_1, \ldots, v_k \in V_k$ be a k-clique in G and let $e_{i,j}$ be the edge between v_i and v_j. We claim that the set $S := K \cup \{v_i : 1 \leq i \leq k\} \cup \{e_{i,j} : 1 \leq i < j \leq k\}$ induces a strongly connected subgraph $G'[S]$; note that $|S| = \ell$. For every $1 \leq i < j \leq k$, the directed closed walk $xv_i w_{e_{i,j}} y_{i,j} w_{e_{i,j}} v_j x$ uses only vertices of S. Moreover, these $\binom{k}{2}$ directed closed walks cover every vertex of S, hence $G'[S]$ is strongly connected.

For the reverse direction, let S be a set of size at most ℓ such that $K \subseteq S \subseteq V(G')$ and $G'[S]$ is strongly connected. Clearly, such a set has to include at least one outneighbor of each vertex in K, which means that S contains at least one vertex of $E'_{i,j}$ for every $1 \leq i < j \leq k$. All the inneighbors of $E'_{i,j} \cup \{y_{i,j}\}$ are in V_i, while all the outneighbors are in V_j, hence each of V_i and V_j has to contain at least one vertex of the solution. Taking into account the quota of $\ell = |K| + k + \binom{k}{2}$ on the size of the solution, this is only possible if S contains exactly one vertex $v_i \in V_i$ for every $1 \leq i \leq k$ and exactly one vertex $w_{e_{i,j}} \in E'_{i,j}$ for every $1 \leq i < j \leq k$. We claim that $v_1 \in V_1$, $\ldots$, $v_k \in V_k$ is a k-clique in G and $e_{i,j}$ is the edge between v_i and v_j. Suppose that v_i is not an endpoint of $e_{i,j}$. Then v_i is not an inneighbor of $w_{e_{i,j}}$. The only other inneighbor of $w_{e_{i,j}}$ is $y_{i,j}$ and the only inneighbor of $y_{i,j}$ in S is $w_{e_{i,j}}$ itself. Therefore, the strongly connected component of $G'[S]$ containing $w_{e_{i,j}}$ consists of only $y_{i,j}$ and $w_{e_{i,j}}$, a contradiction. The argument is similar if v_j is not an endpoint of $e_{i,j}$. Therefore, $e_{i,j}$ is the edge between v_i and v_j, hence $\{v_1, \ldots, v_k\}$ indeed induces a clique in G. □

Note that the reduction in Theorem 13.33 creates a directed graph containing bidirected edges, that is, there are pairs of vertices that are connected by directed edges in both directions. Exercise 13.26 asks for a reduction that does not create bidirected edges.

Exercises

13.1 (🪐). Is there a parameterized reduction from Vertex Cover to Independent Set?

13.2 (🪐). Give a parameterized reduction from Set Cover to Dominating Set.

13.3 (🪐). In the Multicolored Biclique problem the input consists of a bipartite graph G with bipartition classes A, B, an integer k, a partition of A into k sets $A_1, A_2, \ldots, A_k$, and a partition of B into k sets $B_1, B_2, \ldots, B_k$; the question is whether there exists a subgraph of G isomorphic to the biclique $K_{k,k}$, with one vertex in each of the sets A_i and B_i. Prove that Multicolored Biclique is W[1]-hard.

13.4. Prove Theorem 13.12.

13.5. In this exercise we will work out an alternative, purely combinatorial construction of k-paradoxical tournaments of size $2^{\mathrm{poly}(k)}$.

- Find a 2-paradoxical tournament T^* on 7 vertices.
- Assume we are given some tournament T. Construct a tournament T' as follows. The vertices of T' are triples of vertices of T, i.e., $V(T') = V(T) \times V(T) \times V(T)$. Let us consider a pair of triples $u_1 = (a_1, b_1, c_1)$ and $u_2 = (a_2, b_2, c_2)$, where $a_1 \neq a_2$, $b_1 \neq b_2$, and $c_1 \neq c_2$. Consider now pairs $\{a_1, a_2\}$, $\{b_1, b_2\}$, $\{c_1, c_2\}$, and count for how many of them the edge in T was directed from the vertex with subscript 1 to the vertex of subscript 2 (e.g., from a_1 to a_2). If for at least 2 pairs this was the case, we put $(u_1, u_2) \in E(T')$, and otherwise we put $(u_2, u_1) \in E(T')$. For all the pairs of triples where at least one of the coordinates is the same, we put the edge between the triples arbitrarily. Prove that if T was k-paradoxical, then T' is $\lfloor \frac{3k}{2} \rfloor$-paradoxical.
- Define the sequence $T_0, T_1, T_2, \ldots$ as follows: $T_0 = T^*$, and for $m \geq 1$ the tournament T_m is constructed from T_{m-1} using the construction from the previous point. Prove that $|V(T_m)| = 7^{3^m}$ and that T_m is $g(m)$-paradoxical for a function $g(m) \in \Omega((3/2)^m)$.
- Using the previous point, provide an algorithm that, given an integer k, constructs a k-paradoxical tournament of size $2^{\mathcal{O}\left(k^{\log_{3/2} 3}\right)} \leq 2^{\mathcal{O}(k^{2.71})}$. The construction should work in time polynomial with respect to the size of the constructed tournament.

13.6. Let $\mathcal{C}_{s,t,d}$ be the class of circuits having depth at most d where every path from an input node to the output node contains at most t nodes with indegree larger than s. (Thus the class $\mathcal{C}_{t,d}$ defined in Section 13.3 is $\mathcal{C}_{2,t,d}$). Show that, for every $t, d \geq 1$, $s \geq 2$, there is a parameterized reduction from WCS$[\mathcal{C}_{s,t,d}]$ to WCS$[\mathcal{C}_{t,ds}]$. Conclude that replacing $\mathcal{C}_{t,d}$ in Definition 13.16 with $\mathcal{C}_{s,t,d}$ for some fixed $s \geq 2$ would not change the definition of W$[t]$.

13.7. Consider the Weighted Circuit Satisfiability problem, restricted to *monotone* (i.e., without negations) circuits from the class $\mathcal{C}_{1,d}$ for some fixed constant d. Prove that this problem is FPT when parameterized by k, the weight of the assignment in question.

13.8. Give a parameterized reduction from Independent Set to Short Turing Machine Acceptance.

13.9. Consider a restricted variant of the Short Turing Machine Acceptance problem, where we consider only instances (M, x, k) with x being an empty string. Reduce the general Short Turing Machine Acceptance problem to the restricted one.

13.10. Work out the details of the parameterized reduction from Partial Vertex Cover to Short Turing Machine Acceptance (Theorem 13.24).

13.11. Prove that Dominating Set remains W[2]-hard even if restricted to graphs excluding the star $K_{1,4}$ as an induced subgraph.

13.12. Give a parameterized reduction from Dominating Set to Dominating Set with Pattern.

13.13 (☠). Modify the reduction in the proof of Theorem 13.26 to obtain a parameterized many-one reduction from Connected Dominating Set to Dominating Set.

13.14. Given a graph G and an integer k, the Induced Matching problem asks for an induced matching of size k, that is, k edges $x_1y_1, \ldots, x_ky_k$ such that the $2k$ endpoints are all distinct and there is no edge between $\{x_i, y_i\}$ and $\{x_j, y_j\}$ for any $i \neq j$. Prove that Induced Matching is W[1]-complete.

13.15. Given a graph G and an integer k, the Independent Dominating Set problem asks for a set of exactly k vertices that is both an independent set and a dominating set. Prove that Independent Dominating Set is W[2]-complete.

13.16. In the Long Induced Path problem the input consists of a graph G and an integer k, and the question is whether G contains the path on k vertices as an induced subgraph. Prove that this problem is W[1]-hard when parameterized by k.

13.17. Consider the following variant of the Steiner Tree problem: We are given a graph G, a set of terminals $K \subseteq V(G)$, and an integer parameter k. The question is whether one can find a set $X \subseteq V(G) \setminus K$ of cardinality at most k such that $G[K \cup X]$ is connected. In other words, we parameterize the Steiner Tree by the number of nonterminals that can be added to the tree, while the number of terminals can be unbounded. Prove that this parameterization of Steiner Tree is W[2]-hard.

13.18 (🖉). Is List Coloring W[1]-hard parameterized by the vertex cover number?

13.19. Give a parameterized reduction from Exact Odd Set to Exact Even Set.

13.20 (☠). Show that Unique Hitting Set and Exact Unique Hitting Set are in W[1].

13.21. Show that the graph versions of all five problems in Theorem 13.32 are W[1]-hard. (The graph version of Exact Unique Hitting Set is called Perfect Code.)

13.22. In the Subset Sum problem the input consists of a sequence of n integers $a_1, a_2, \ldots, a_n$ and two integers s, k, and the question is whether one can find a set $I \subseteq \{1, 2, \ldots, n\}$ of exactly k indices such that $\sum_{i \in I} a_i = s$. Prove that Subset Sum is W[1]-hard when parameterized by k.

13.23. In the Set Packing problem the input consists of family $\mathcal{F}$ of subsets of a finite universe U and an integer k, and the question is whether one can find k pairwise disjoint sets in $\mathcal{F}$. Prove that Set Packing is W[1]-hard when parameterized by k.

13.24. In the Exact CNF-SAT problem the input consists of formula φ in the conjunctive normal form and an integer k, and the question is whether one can find an assignment ψ of weight exactly k such that in each clause of φ exactly one literal is satisfied. Prove that Exact CNF-SAT is W[1]-hard when parameterized by k.

13.25. In the Permutation Composition problem the input consists of a family $\mathcal{P}$ of permutations of a finite universe U, additional permutation π of U, and an integer k, and the question is whether one can find a sequence $\pi_1, \pi_2, \ldots, \pi_k \in \mathcal{P}$ such that $\pi = \pi_1 \circ \pi_2 \circ \ldots \circ \pi_k$. Prove that Permutation Composition is W[1]-hard when parameterized by k.

13.26 (🪐). Modify the reduction of Theorem 13.33 in such a way that the constructed graph has no bidirected edges.

13.27. In the Vertex k-Way Cut problem the input consists of a graph G and two integers k, s, and the question is whether one can find a set X of at most k vertices such that the graph $G - X$ has at least s connected components. Prove that Vertex k-Way Cut is W[1]-hard when parameterized by $k + s$.

13.28. Given a bipartite graph G with bipartite classes $A, B \subseteq V(G)$ and an integer k, the Hall Set problem asks for a *Hall set* of size at most k, that is, a set $S \subseteq A$ of size at most k such that $|N(S)| < |S|$. Show that Hall Set is W[1]-hard.

13.29 (☠). Recall that a graph is 2-degenerate if its every subgraph contains a vertex of degree at most 2. In the 2-degenerate Vertex Deletion problem one is given a graph G and an integer k, and the question is whether there exists a set X of at most k vertices such that the graph $G - X$ is 2-degenerate. Prove that 2-degenerate Vertex Deletion is W[2]-hard when parameterized by k (if you find it more convenient, you may prove just W[1]-hardness).

13.30 (☠). In the Longest Common Subsequence problem the input consists of a finite alphabet Σ, a set of strings $s_1, s_2, \ldots, s_\ell \in \Sigma^*$ and an integer k, and the question is whether there exists a string $s \in \Sigma^*$ of length k such that s is a subsequence of s_i for every $1 \leq i \leq s$. Prove that Longest Common Subsequence is W[1]-hard when parameterized by k.

Hints

13.1 The following algorithm satisfies the formal definition of a parameterized reduction: solve the Vertex Cover instance in FPT time and output a trivial yes-instance or no-instance of Independent Set. More generally, if A is FPT, then A has a parameterized reduction to any parameterized problem B that is nondegenerate in the sense that it has at least one yes-instance and at least one no-instance.

13.4 Sample the tournament by choosing the orientation of each edge independently and uniformly at random. Prove that for $n \geq r(k)$ the sampled tournament is k-paradoxical with positive probability.

13.6 Every OR-node or AND-node with indegree at most s can be simulated with a chain of at most $s - 1$ nodes with indegree 2. This way, we can construct an equivalent circuit such that depth increases only by a factor of at most s. In fact, by replacing a node with indegree s with an (almost) complete binary tree of indegree-2 nodes, we can restrict the increase of depth to $\mathcal{O}(\log s)$.

13.7 Observe that there is only a constant number of large gates. Guess the output of all large gates. For each large OR gate, guess which in-edge of this gate is satisfied, and what is the assignment of the input gates that make this particular in-edge satisfied (it depends only on a constant number of input gates). Finally, to resolve large AND-gates, design a simple branching algorithm.

13.8 In the first k steps, the Turing machine guesses k vertices and writes it on the tape. In the next $\binom{k}{2}$ phases, the Turing machine checks whether every pair of the guessed vertices are distinct and nonadjacent. Each phase can be implemented in $\mathcal{O}(k)$ steps.

13.9 Modify the Turing machine in such a way that it writes x on the tape in the first $|x|$ steps (and then the head returns to the starting position).

13.11 Try to adjust the reduction for Connected Dominating Set of Theorem 13.15 so that the output graph has no $K_{1,4}$ as an induced subgraph. You would probably want to turn as many big parts of the graph into cliques as possible.

13.12 The reduction of Theorem 13.15 from Dominating Set to Connected Dominating Set has the property that if the created instance has a connected dominating set of size k, then it has a dominating set of size k inducing a clique.

13.13 Express Connected Dominating Set as $2^{\mathcal{O}(k^2)}$ instances of Dominating Set with Pattern and reduce them to $2^{\mathcal{O}(k^2)}$ instances of Dominating Set (or Set Cover if you will). Then use the composition algorithm of Section 15.2.3 to create a single instance of Set Cover.

13.16 Reduce from Multicolored Independent Set. Given a Multicolored Independent Set instance $(G, k, (V_1, V_2, \ldots, V_k))$ aim at an instance (G', k') of Long Induced Path with $k' = 3k$ and, in the intended solution, the $(3i-1)$-th vertex of the path belongs to V_i, whereas the remaining vertices belong to some gadgets you introduce.

13.17 Reduce from Set Cover.

13.18 Note that treewidth of a graph is at most its vertex cover number. Therefore, the statement of Theorem 13.30 (W[1]-hardness parmeterized by treewidth) *does not* imply W[1]-hardness parameterized by vertex cover number, as vertex cover can be a much larger parameter. However, by looking at the proof of Theorem 13.30, we can observe that the reduction creates instances of List Coloring with vertex cover number at most k, which means that the same proof proves also the (stronger) result that the problem is W[1]-hard parameterized by the vertex cover number.

13.19 By introducing a new element x and a new set $\{x\}$ on it, we may assume that the parameter k of the Exact Odd Set instance is even. To create an instance of Exact Even Set, let us introduce a new element y, add it to every set, let us introduce a new set $U \setminus \{y\}$, and set $k' := k + 1$ (which is an odd number). Now the set $U \setminus \{y\}$ ensures that every solution of size exactly k' contains y. Consequently, if A is a set of the Exact Odd Set instance, then the fact that the solution of the Exact Even Set instance has even intersection with $A \cup \{y\}$ implies that it has odd intersection with A.

13.20 We can prove membership by a reduction to Short Turing Machine Acceptance. The Turing machine first guesses the (at most k) elements of the solution. Then by testing all the $\binom{k}{2}$ pairs, it verifies that there is no set in the instance that contains two elements of the solution. Finally, to check that all the sets are hit by the solution, the Turing machine sums the total degree of the elements (that is, the number of sets they are contained in) of the solution. As no set is hit by more than one element of the solution, this sum is exactly the number of sets hit by the solution, hence it is exactly the number of sets hit by the solution. To implement this algorithm in a Turing machine, we have to

hard-code which pairs of elements are contained in some set together and the degree of each element.

13.22 Reduce from Perfect Code or Exact Unique Hitting Set.

13.23 Reduce from Independent Set: for each vertex v of the input instance (G, k), create a set F_v consisting of all edges of G incident to v.

13.24 Reduce from Perfect Code or Exact Unique Hitting Set.

13.25 Reduce from Perfect Code or Exact Unique Hitting Set.

13.26 Split each nonterminal vertex v into two vertices v_{in} and v_{out} connected by an edge $(v_{\text{in}}, v_{\text{out}})$. Modify k' appropriately.

13.27 Try a reduction similar to the one given in the proof of Theorem 13.29.

13.28 Reduction from Clique. Given a graph G, we construct a bipartite graph where class A corresponds to the edges of G and class B corresponds to the vertices of B; the vertex of A corresponding to edge uv of G is adjacent to the two vertices $u, v \in B$. Additionally, we introduce a set of $\binom{k}{2} - k - 1$ vertices into B and make them adjacent to every vertex of A. Show that every Hall set of size at most $\binom{k}{2}$ has size exactly $\binom{k}{2}$ and corresponds to the edges of a k-clique in G.

13.29 You may find the following characterization of d-degenerate graphs useful: a graph is d-degenerate if one can delete all the vertices of the graph one by one, each time removing a vertex that at the current moment has degree at most d.

Start the reduction from Set Cover. Create k set-choice gadgets that emulate choices of k sets of the solution. Each gadget should be 3-regular, and should become 2-degenerate after removing any of its vertices. Then create a gadget for each element of the universe, and wire these gadgets with the set-choice gadgets to encode the input instance. Ensure the following properties: (i) If an element is not covered, then the closed neighborhood of the corresponding gadget induces a 3-regular graph untouched by the solution. (ii) If an element is covered, then the closed neighborhood of the corresponding gadget is affected by the solution and it is possible to "rip" the whole gadget in the process described in the previous paragraph.

13.30 Reduce from Multicolored Clique. Let $(G, k, (V_1, V_2, \ldots, V_k))$ be a Multicolored Clique instance. The string s should encode the choice of vertices and edges of the clique, that is, $V(G) \cup E(G) \subseteq \Sigma$ (you may need some additional special symbols in Σ that will serve as separators or markers). To this end, fix an arbitrary total order $\preceq$ on each set V_i. Let $V_i^{\uparrow}$ be the string consisting of all vertices of V_i in the increasing order of $\preceq$, and $V_i^{\downarrow}$ be the string $V_i^{\uparrow}$ reversed. The orders on V_i and V_j impose a lexicographical order on each set $E_{i,j}$ of edges connecting V_i and V_j in the graph G, and hence the strings $E_{i,j}^{\uparrow}$ and $E_{i,j}^{\downarrow}$ are defined in a natural manner. Consider two strings

$$s_1 = V_1^{\uparrow} V_2^{\uparrow} \ldots V_k^{\uparrow} E_{1,2}^{\uparrow} E_{1,3}^{\uparrow} \ldots E_{k-1,k}^{\uparrow}$$
$$s_2 = V_1^{\downarrow} V_2^{\downarrow} \ldots V_k^{\downarrow} E_{1,2}^{\downarrow} E_{1,3}^{\downarrow} \ldots E_{k-1,k}^{\downarrow}.$$

Observe that the longest common subsequence of s_1 and s_2 is of length $k + \binom{k}{2}$ and contains exactly one vertex v_i from each set V_i and exactly one edge $e_{i,j}$ from each set $E_{i,j}$.

Now, design a "gadget" consisting of two strings that ensures that v_i is an endpoint of $e_{i,j}$, for each $1 \leq i < j \leq n$. Similarly, verify the second endpoint of $e_{i,j}$.

Bibliographic notes

Downey and Fellows [149, 150, 148] defined the notion of parameterized reductions and recognized that they can be used to transfer fixed-parameter intractability results. The W[1]-hardness of Clique and Independent Set on regular graphs, as well as the fact that this can be used to show that Partial Vertex Cover is W[1]-hard, was observed by Cai [64] and Marx [349]. The introduction of Multicolored Clique and its W[1]-hardness is usually attributed to Fellows, Hermelin, Rosamond, and Vialette [179], but earlier Pietrzak [383] studied the same problem under the name Partitioned Clique.

The W[2]-completeness of Dominating Set on Tournaments was proved by Downey and Fellows [152]; however, the issue of how to construct efficiently a k-paradoxical tournament was glossed over. The probabilistic proof of the existence of small k-paradoxical tournaments, i.e., Theorem 13.12, was given by Erdős [166]. The best known deterministic construction, given by Graham and Spencer [233], achieves size $\mathcal{O}(4^k \cdot k^2)$ for a k-paradoxical tournament and is based on number-theoretical concepts. The combinatorial construction of Exercise 13.5 has been proposed by Tyszkiewicz [421]. The vast majority of parameterized reductions in the literature are actually polynomial-time reductions. The reduction to Dominating Set on Tournaments presented in Theorem 13.14 is one of the rare exceptions: the running time depends superpolynomially on k. Other examples of such reductions can be found in the W[1]-hardness proofs of the Vapnik-Chervonenkis (VC) dimension [152] and certain parameterizations of Closest Substring [348] (see Theorem 14.27 in Section 14.4).

The W-hierarchy was defined by Downey and Fellows [149, 148, 149, 152, 150, 151]. These series of papers prove, among other results, the W[1]-completeness of Independent Set (Theorem 13.18) and the characterization of W[t] by normalized circuit satisfiability (Theorem 13.19). The connection of W[1] and Turing machines was investigated by Cai, Chen, Downey, and Fellows [67]. They show W[1]-hardness of Short Turing Machine Acceptance by a reduction from Clique and show membership in W[1] by reducing Short Turing Machine Acceptance to Weighted Circuit Satisfiability with weft-1 circuits. Together with the W[1]-hardness of Independent Set (Theorem 13.18), this gives a reduction from Short Turing Machine Acceptance to Independent Set. In Section 13.22, we worked out a direct reduction from Short Turing Machine Acceptance to Independent Set to give self-contained evidence (without the need for the W-hierarchy and Theorem 13.18) why Independent Set is unlikely to be fixed-parameter tractable.

Cesati [72, 71] observed that reduction to Short Turing Machine Acceptance can be a convenient way of proving membership in W[1]. We used this technique for Partial Vertex Cover (Theorem 13.24) and for Unique Hitting Set (Exercise 13.20).

The W[1]-hardness proof of Balanced Vertex Separator is by Marx [346]. The W[1]-hardness proof of List Coloring parameterized by treewidth is by Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, and Thomassen [177]. In Section 14.29, we will see a different way of proving W[1]-hardness for List Coloring parameterized by treewidth, which also works for planar graphs.

Downey and Fellows [151] proved the W[1]-hardness of Perfect Code. Downey, Fellows, Vardy, and Whittle [155] used the W[1]-hardness of Perfect Code to show (via complicated reductions) the W[1]-hardness of variants of Odd Set (and of further problems that we do not discuss here). In Section 13.6, we presented a very simple and self-contained W[1]-hardness proof for Odd Set that can be adapted for other variants.

The W[2]-hardness of Dominating Set in graphs excluding $K_{1,4}$ as an induced subgraph has been proved independently by Cygan, Philip, Pilipczuk, Pilipczuk, and Wojtaszczyk [119] and Hermelin, Mnich, van Leeuwen, and Woeginger [259]. It is worth mentioning that, as proved in both of these papers, Dominating Set becomes FPT if restricted to graphs excluding $K_{1,3}$ as an induced subgraph (i.e., in claw-free graphs).

The W[1]-hardness of various edge and vertex versions of Strongly Connected Steiner Subgraph was shown by Guo, Niedermeier, and Suchý [244] and by Chitnis, Hajiaghayi, and Marx [92]. In Section 13.6, we have presented a somewhat simpler proof that works only for the vertex variant.

Vertex deletion problems to d-degenerate graphs (Exercise 13.29) were studied by Mathieson [359]. It appears that they are even harder than as stated in Exercise 13.29.

Given a graph G and an integer k, the Biclique problem asks for a $K_{k,k}$ subgraph, that is, two disjoint sets $X, Y \subseteq V(G)$ vertices of size exactly k such that every vertex of X is adjacent to every vertex of Y. The fixed-parameter tractability of Biclique was a longstanding open question. Very recently, Lin [321] proved that Biclique is W[1]-hard, resolving this question. (Exercise 13.3 considers the multicolored version of the problem, which can be shown to be W[1]-hard in a much easier way.)