University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Marcin Pilipczuk

# New techniques applicable to selected NP-hard problems

*PhD dissertation*

Supervisor
dr hab. Łukasz Kowalik

Institute of Informatics
University of Warsaw

September 2011

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

| | |
|---|---|
| September 21, 2011 | ..... |
| *date* | *Marcin Pilipczuk* |

Supervisor's declaration:
the dissertation is ready to be reviewed

| | |
|---|---|
| September 21, 2011 | ..... |
| *date* | *dr hab. Łukasz Kowalik* |

**Streszczenie.** Każdy NP-zupełny problem można rozwiązać w czasie wykładniczym przez przejrzenie całej przestrzeni możliwych rozwiązań. Dziedzina algorytmów umiarkowanie wykładniczych zajmuje się poszukiwaniem sposobów dokładnego rozwiąznywania problemów NP-zupełnych asymptotycznie szybciej niż stosując rozwiązania naiwne. Zaskakująco często takie algorytmy istnieją, a ich otrzymanie i analiza wymaga nowych, ciekawych obserwacji dotyczących rozważanego problemu.

W pierwszej części tej rozprawy podajemy algorytmy dla trzech problemów: Capacitated Dominating Set, Minimum Maximal Irredundant Set i pewnego problemu szeregowania zadań. We wszystkich trzech przypadkach istnieją proste rozwiązania działające w czasie $O(2^n)$ i nasze algorytmy przełamują tę naturalną barierę. Wszystkie te rezultaty są oparte o nowe obserwacje pozwalające ograniczyć rozmiar przestrzeni rozwiązań.

W złożoności parametryzowanej zakładamy, że z każdym egzemplarzem $I$ rozważanego (zazwyczaj NP-zupełnego) problemu stowarzyszony jest parametr $k$. Poszukujemy algorytmów (zwanych parametryzowanymi) działających w czasie $f(k)|I|^{O(1)}$ dla pewnej obliczalnej funkcji $f$. Innymi słowy, chcemy, aby ponadwielomianowy czynnik w złożoności czasowej, prawdopodobnie nieunikniony w przypadku problemu NP-zupełnego, zależał tylko od parametru $k$. Szczególnym przypadkiem algorytmów parametryzowanych jest dziedzina kernelizacji, która zajmuje się poszukiwaniem i analizą algorytmów przetwarzających dany egzemplarz problemu do równoważnego, którego wielkość zależy tylko od wartości parametru.

Druga część tej rozprawy jest poświęcona dwóm wynikom w dziedzinie złożoności parametryzowanej. Pokazujemy pierwszy znany algorytm parametryzowany dla problemu Subset Feedback Vertex Set, blisko związanego z grafowymi problemami separacyjnymi, grającymi obecnie kluczową rolę w złożoności parametryzowanej. Ponadto dowodzimy, że jeżeli hierarchia wielomianowa nie ustala się na trzecim poziomie, wiele problemów grafowych zawierających wymóg spójności nie posiada algorytmów kernelizacyjnych, w których rozmiar wynikowego egzemplarza jest zależny wielomianowo od parametru, nawet przy ograniczeniu się do grafów o ograniczonej degeneracji. Ten wynik pokazuje, że znane algorytmy kernelizacyjne dla grafów o wykluczonym minorze (będących podklasą grafów o ograniczonej degeneracji) w istotny sposób wykorzystują topologiczne własności tych klas grafów.
**Słowa kluczowe:** algorytmy umiarkowanie wykładnicze, algorytmy parametryzowane, kernelizacja, bariera $2^n$, grafy o ograniczonej degeneracji.
**Klasyfikacja tematyczna ACM:** F.2.2, G.2.1, G.2.2.

**Abstract.** All NP-complete problems can be solved in exponential time by enumerating the space of possible solutions. In the area of moderately-exponential algorithms, we seek for exact algorithms for NP-complete problems that are faster than the naive ones. Surprisingly, often such results exist and their development leads to a good insight into a considered problem.

In the first part of this dissertation we give algorithms for three problems: Capacitated Dominating Set, Minimum Maximal Irredundant Set and one job scheduling problem. In all cases there exists a simple $O(2^n)$-time algorithm, and our results break the natural $2^n$-barrier. All three results are based on new observations on the considered problems that allow us to limit the search space.

In the parameterized complexity setting we assume that a given instance $I$ (of a usually NP-complete problem) comes up with a parameter $k$. We seek for algorithms (called fixed-parameter algorithms) working in time $f(k)|I|^{O(1)}$ for some computable function $f$. In other words, we want the superpolynomial factor in the time complexity, probably unavoidable for NP-complete problems, to depend on the parameter only. A special case of fixed-parameter algorithms is the field of kernelization, where one seeks for a polynomial-time preprocessing algorithms that shrink a given instance to one with size bounded by a function of the parameter $k$.

The second part of this dissertation is devoted to two results in this fields. First, we show the first known fixed-parameter algorithm for Subset Feedback Vertex Set, a problem closely related to graph-cutting problems that are now central in parameterized complexity. Second, we prove that, unless the polynomial hierarchy collapses up to its third level, many problems involving connectivity requirement do not admit a kernelization algorithm with polynomial (in the parameter) guarantee on the output size, even if the input graph is restricted to be of bounded degeneracy. This proves that known kernelization algorihtms for graphs excluding a fixed minor (being a subclass of graphs of bounded degeneracy) indeed require the topological properties of these graph classes.

**Keywords:** moderately-exponential algorithms, fixed-parameter algorithms, kernelization, $2^n$-barrier, bounded degeneracy graphs.
**ACM classification:** F.2.2, G.2.1, G.2.2.

# Chapter 1

# Introduction

Many real-life combinatorial problems turn out to be NP-hard, and thus probably not solvable efficiently, that is, in polynomial time. Since the discovery of the NP-hardness phenomenon in late 1960s and early 1970s, researchers were developing different methods of coping with hard problems. One of the classical approach are approximation algorithms, where one weakens the requirement that the solution must be optimal and asks for one close to the optimal. However, the discovery of the PCP theorem and subsequent work show that many important problems are hard to approximate, including the Clique problem [89].

Thus, in some applications, polynomial-time approximation is not sufficient and we may ask what we can gain by allowing superpolynomial time. Naturally, all problems in NP can be solved in exponential time by trying all possible solutions, whose sizes are bounded polynomially in the input size. However, it turns out that in many cases there exist much faster (but still superpolynomial) algorithms solving NP-hard problems optimally. We call such algorithms *moderately-exponential*. One of the oldest examples is the $O(2^n n^2)$-time dynamic programming algorithm of Held and Karp (1962) solving Traveling Salesman Problem [94]. Moderately-exponential algorithms attracted a lot of attention in the last decade, and the race for fastest algorithms solving problems like Clique, Dominating Set, Chromatic Number or Hamiltonian Cycle lead to the development of new techniques such as Measure&Conquer [73], Fast Subset Convolution and usages of the inclusion-exclusion principle [11, 13] or usages of the polynomial identity testing [10]. For more background of moderately-exponential algorithms we refer the reader to the recent textbook of Fomin and Kratsch [75].

In the real-life applications the input instances are usually somewhat specific and it turns out that many heuristic algorithms, even with horrendous worst-case running time, behave reasonably. *Parameterized complexity* is a mathematical framework that allows rigorous analysis of such algorithms. For a given problem, each input instance $x$ is accompanied with a parameter $k$ and we seek for algorithms (called fixed parameter algorithms) running in time $f(k)|x|^{O(1)}$ for some computable (usually exponential) function $f$. Intuitively, we try to encapsulate the exponential explosion in the running time, probably unavoidable in NP-hard problems, in the function of only the parameter, not the size of the input instance. Thus, if the parameter is small, the algorithm runs quickly. Typical examples of parameters include solution size (e.g., 'is there a vertex cover of size at most $k$ in the input graph $G$?'), or some structural parameters of the input (e.g., 'given a graph of treewidth $k$, find a minimum vertex cover').

A very special way of obtaining parameterized algorithms is the idea of *kernelization*. A kernelization algorithm reduces in polynomial time the input instance $x$ with parameter $k$ to an equivalent one with the size bounded by $g(k)$ for some computable function $g$. Thus, kernelization can be seen as a way of preprocessing and shrinking the input instance, so that the algorithms applied later run faster. Note that if we solve the problem for the reduced instance by an exhaustive search algorithm, we obtain a fixed-parameter algorithm for any instance. We usually aim at polynomial kernels, that is, with function $g$ being a polynomial.

The parameterized complexity framework was very successful. As discussed in the introduction to the textbook of Downey and Fellows [56], it turns out that many graphs appearing in real-life applications have small treewidth, thus allowing efficient dynamic programming algorithms on tree-width decomposition. A lot of applications come from computational biology. For example, the DNA samples form a graph close to an interval one, thus it makes sense to parameterize by the edition distance to interval graphs. From theoretical point of view, the research in parameterized complexity lead to a development of many important techniques, including color coding [5], iterative compression [120] or bidimensionality [53, 77, 78]. The $W$-hierarchy, discovered in 1990s, shows that some problems, including Clique and Dominating Set parameterized by the solution size, probably do not admit fixed parameter algorithms. In 2008, Fortnow and Santhanam [79] and Bodlaender et al. [15] developed a way to show hardness of polynomial kernelization. Today parameterized complexity is a big and active research area. For more

background we refer to the textbooks by Downey and Fellows [56], Flum and Grohe [71] and Niedermeier [114].

In this dissertation we focus on results that all can be classified as *barrier-crossing* ones, where a new algorithm crosses some hardness barrier that was considered tough or even unlikely to overcome. Such results are very valuable in algorithmics, as usually they significantly change the place of the considered problem in the complexity hierarchy. This type of results includes all proofs that a problem belongs to a certain (the smaller, the better) well-known complexity class, such as LOGSPACE (problems that admit logarithmic space algorithms) or FPT (problems that admit fixed-parameter algorithms). However, in many cases also crossing some barrier in the time or space complexity makes a small revolution, too. For example, a recent $O(n \log\log n)$-time algorithm for finding minimum cuts in planar graphs [96] surprisingly improved upon $O(n \log n)$ algorithm by Reif [121] from 1983. The Reif's result was previously considered optimal, as $O(n \log n)$ complexity seems natural for this problem.

We would like to note that all hardness results in algorithmics can be seen as statements that in a considered problem one particular barrier is impossible (or very unlikely) to be overcome. Thus, the barrier-crossing results can be seen as complementary to the hardness ones.

The first part of this dissertation is devoted to one specific barrier that occurs in moderately-exponential algorithms. In many cases, a brute-force or natural dynamic programming approach leads to an algorithm running in time $2^n n^{O(1)}$ and it seems hard to break the $2^n$ barrier, i.e., to obtain an algorithm running in time $O(c^n)$ for some constant $c < 2$. A successful stories in this line of research include Dominating Set [73] and Hamiltonian Cycle in undirected graphs [10], but for many problems, including other variants of Hamiltonian Cycle or TSP as well as Chromatic Number, the question of the existence of an algorithm faster than $2^n n^{O(1)}$ remains widely open. The hardness of one particular problem — the question of satisfiability of boolean formulae on $n$ variables — conjectured as Strong Exponential Time Hypothesis by Impagliazzo and Paturi [95] — is used as an argument that other problems are hard [41, 42, 108, 115].

In this dissertation we present algorithms that break the $2^n$ barrier for three problems: Capacitated Dominating Set, Minimum Maximal Irredundant Set and a job scheduling problem, denoted $1|\text{prec}|\sum C_i$ in the Graham notation. In all cases, the algorithms are based on some interesting and non-trivial observations that allow us to limit the size of the search

space. All algorithms, together with precise problem definitions, are gathered in Chapter 2.

In the second part (Chapter 3) of this dissertation we present a few results from the area of parameterized complexity. The first and the main one is a fixed-parameter algorithm for Subset Feedback Vertex Set, a problem closely related to Feedback Vertex Set, Multiway Cut and Multicut, three problems that play central roles in parameterized complexity now. By this result we show that Subset Feedback Vertex Set crosses the barrier of being fixed-parameter tractable. The second half of Chapter 3 is devoted to hardness of polynomial kernelization for a few connectivity problems on bounded degeneracy graphs. As discussed before, such results show that in this case the barrier of admitting a polynomial kernel is highly unlikely to be crossed.

The results in this dissertation are included in the following conference and journal papers.

1. The algorithm solving Capacitated Dominating Set, described in Section 2.2, was presented at SWAT 2010 [47]; the journal version is published in Information Processing Letters [49].

2. The algorithm solving Minimum Maximal Irredundant Set, described in Section 2.3, was presented at CIAC 2010 [48] and is included in a joint journal paper with another group of researchers that solved the problem independently [9].

3. The algorithm solving $1|\text{prec}|\sum C_i$, described in Section 2.4, was presented at ESA 2011 [44].

4. The fixed-parameter algorithm solving Subset Feedback Vertex Set, described in Section 3.2, was presented at ICALP 2011 [45].

5. The proofs of the hardness of polynomial kernelization of connectivity problems in degenerate graphs, described in Section 3.4, were presented at WG 2010 [43].

**Acknowledgements.** I would like to thank my co-workers at University of Warsaw, Marek Cygan, Michał Pilipczuk and Jakub Onufry Wojtaszczyk, for all the effort that lead to the results included in this dissertation. I would also like to thank here Dominik Scheder for some useful discussions on the

$1|\text{prec}|\sum C_i$ problem during his stay in Warsaw in April 2011. Moreover, I am extremely grateful to my advisor, Łukasz Kowalik, for many helpful comments on presentation and write-up. Finally, I would like to thank my family: without their support this dissertation would not be done in time.

## 1.1 Notation

All graphs in this dissertation are undirected and finite and, unless otherwise specified, simple. For a graph $G$ by $V(G)$ and $E(G)$ we denote the vertex and the edge set, respectively. If the graph is clear from the context, we write only $V$ and $E$. Given a vertex $v \in V(G)$, we define its (open) neighbourhood by $N_G(v) = \{u : uv \in E(G)\}$ and closed neighbourhood by $N_G[v] = N_G(v) \cup \{v\}$. We extend this notation to subsets of vertices: given $X \subseteq V(G)$, we define $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. We omit the subscript whenever the graph is clear from the context. For a set of vertices $X \subseteq V(G)$ by $G[X]$ we denote the subgraphs induced by $X$. For a set $X$ of vertices or edges, by $G \setminus X$ we denote the graph obtained from $G$ by removing the vertices or edges of the set $X$. If $X = \{x\}$, we shorten the notation $G \setminus \{x\}$ to $G \setminus x$. For $F \subseteq E(G)$, by $V(F)$ we denote the set of endpoints of the edges in $F$. We say a vertex $v \in V$ dominates $u \in V$ if $u = v$ or $uv \in E$ (i.e., a vertex dominates its closed neighbourhood). We say that a set $W$ dominates a vertex $u$ if some vertex $v \in W$ dominates $u$.

A cycle in $G$ is a sequence of vertices $v_1 v_2 \dots v_m \in V$ such that $v_i v_{i+1} \in E$ and $v_m v_1 \in E$. We say a cycle is simple if $m > 2$ and the vertices $v_i$ are pairwise different. If we consider multigraphs (i.e., graphs with multiple edges and loops), a simple cycle can have two vertices if there is a multiple edge between them, or a single vertex if there is a loop attached to it. We call an edge $vw \in E$ a bridge if in $(V, E \setminus \{vw\})$ the vertices $v$ and $w$ are in different connected components. Note that no simple cycle can contain a bridge as one of its edges. Given subsets $X, Y \subseteq V$, by $E(X, Y)$ we denote the set of edges with one endpoint in $X$ and the other in $Y$.

As we focus on exponential-time algorithms, we neglect the polynomial (in the size of the input) terms in the time and space complexity. We introduce the $O^\star$ notation, which is the big-Oh notation with terms polynomial in the input size suppressed, i.e., $f(n) = O^\star(g(n))$ when $f(n) = O(g(n)n^D)$ for some constant $D$. For example, $2^n n^3 = O^\star(2^n)$. Note that $c^n n^D = O((c + \varepsilon)^n)$ for any constant $c > 1$, $D > 0$ and $\varepsilon > 0$. As the $O^\star$ notation suppresses

terms polynomial in the input size, we can use it also in the context of fixed-parameter algorithms, for example $2^{k^2}n^3 = O^\star(2^{k^2})$ where $k$ is the parameter and $n$ is the number of vertices of the input graph.

# Chapter 2

# Moderately-exponential algorithms

## 2.1 Introduction

Although all NP-hard problems are equivalent from the point of view of polynomial-time transformations, the time complexity of fastest algorithms that find optimal solutions can vary greatly. Moreover, as mentioned in the introduction, in some cases polynomial-time approximation does not give sufficiently good results and we simply need to find optimal solution. The area of moderately-exponential algorithms seeks for fast, in many cases much faster than the naive ones, algorithms that solve NP-hard problems optimally.

We would like to stress two other motivations for investigating exponential complexity of NP-hard problems. First, note that improving the time complexity from $c_1^n$ to $c_2^n$ increases the size of the input for which computation is feasible by a *multiplicative* factor (of $\log c_1 / \log c_2$). This should be compared to the *addictive* factor gained by an improvement of the speed of the hardware.

The second motivation is of different, theoretical type. The design of moderately-exponential algorithms often reveals interesting structural properties of the considered problems that may be of independent interest. All three algorithms presented in this chapter are very interesting from this point of view: although the speed increase is in two cases really small, all algorithms are based on some novel observations about respective problems.

Although moderately-exponential algorithms appear in the literature since

1960s (e.g., the classical dynamic programming algorithm of Held and Karp for TSP [94]), it gained big interest and become an active area of research in the last decade. The most interesting results include the algorithm for Bandwidth of Feige [67], the Measure&Conquer technique and its application for Clique and Dominating Set problem [73], Fast Subset Convolution and its many applications [11, 12, 13, 127], as well as randomized algorithms based on polynomial identity testing [10].

In case of many problems, the best known algorithm, usually brute-force or simple dynamic programing, works in $O^\star(2^n)$ time, where $n$ is some natural measure of the input size, in particular, in case of graph problems, $n$ is always the number of vertices. Going beyond this $2^n$ *barrier* is fairly nontrivial. This was the case of Dominating Set [73] or Hamiltonian Cycle in undirected graphs [10], and currently a question of an algorithm for Chromatic Number or TSP faster than $O^\star(2^n)$ is a major open question in the field. It is even conjectured that the satisfiability of boolean formulae with $n$ variables cannot be solved faster than an exhaustive search in $O^\star(2^n)$ time [95] and this hypothesis is a starting point of a few hardness results [41, 42, 108, 115].

In this dissertation we present algorithms that break the $2^n$ barrier for three problems:

1. In Section 2.2 we present an $O(1.89^n)$ algorithm that solves Capacitated Dominating Set.

2. In Section 2.3 we present an $O(1.999965^n)$ algorithm that solves Minimum Maximal Irredundant Set.

3. In Section 2.4 we present an $O((2-5\cdot 10^{-16})^n)$ algorithm that solves a scheduling problem denoted by $1|\text{prec}|\sum C_i$ in the Graham notation.

The value of our results is not in the speed increase — which is in the last two cases admittedly small — but in the claim that the $2^n$-barrier can be crossed. Moreover, the results give a new insight into the considered problems. In particular, we would like to mention here the core observation and lemma in the algorithm for $1|\text{prec}|\sum C_i$ (Section 2.4.5) as an example of new ideas developed while breaking the $2^n$ barrier.

In a few places in this chapter we use the following simple bound on binomial coefficients.

**Lemma 2.1.** *Let $0 < \alpha < 1$ be a constant. Then*

$$\binom{n}{\alpha n} = O^\star\left(\left(\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}}\right)^n\right).$$

*In particular, if $\alpha \neq 1/2$ then there exists a constant $c_\alpha < 2$ that depends only on $\alpha$ and*

$$\binom{n}{\alpha n} = O^\star\left(c_\alpha^n\right).$$

*Proof.* By Stirling's formula,

$$n! = \Theta\left(\frac{n^n\sqrt{n}}{e^n}\right).$$

Thus

$$\begin{aligned}
\binom{n}{\alpha n} &= \frac{n!}{(\alpha n)!((1-\alpha)n)!} \\
&= \Theta\left(\frac{\frac{n^n\sqrt{n}}{e^n}}{\frac{(\alpha n)^{\alpha n}\sqrt{\alpha n}}{e^{\alpha n}}\frac{((1-\alpha)n)^{(1-\alpha)n}\sqrt{(1-\alpha)n}}{e^{(1-\alpha)n}}}\right) \\
&= \Theta\left(\frac{1}{\alpha^{\alpha n}(1-\alpha)^{(1-\alpha)n}\sqrt{n}}\right) \\
&= O^\star\left(\left(\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}}\right)^n\right)
\end{aligned}$$

The second part of the lemma follows from the AM-GM inequality for values $\frac{1}{\alpha}$ and $\frac{1}{1-\alpha}$ with weights $\alpha$ and $(1-\alpha)$, respectively:

$$\left(\frac{1}{\alpha}\right)^\alpha\left(\frac{1}{1-\alpha}\right)^{1-\alpha} \leq \alpha\cdot\frac{1}{\alpha} + (1-\alpha)\cdot\frac{1}{1-\alpha} = 2.$$

Note that the equality in the AM-GM inequality holds if and only if $\frac{1}{\alpha} = \frac{1}{1-\alpha}$, i.e., $\alpha = 1/2$. □

## 2.2 Capacitated domination

In a graph $G$, a set $S \subseteq V$ is called a *dominating set* if it dominates the whole $V$. The Dominating Set problem asks for the size of the smallest dominating set of $G$, which we denote by $\gamma(G)$.

The Dominating Set problem is one of the most intensively studied problems in algorithmics. It is NP-complete to solve exactly and NP-hard to approximate within a sublogarithmic factor, but a simple greedy algorithm obtains a $(1 + \log |V|)$-approximation. A straightforward algorithm solves Dominating Set in $O^\star(2^n)$ time and at the beginning of this century it was an important open problem in the moderately-exponential algorithm community to break the $2^n$-barrier for this problem. This was independently achieved by Grandoni [85], Fomin et al. [76] and Randerath and Schiermeyer [122]. The search for faster algorithms for the Dominating Set problem led to the discovery of the Measure&Conquer technique [72, 73]. This technique helps in obtaining good bounds on the size of a search tree of backtracking algorithms. The currently fastest algorithm solving Dominating Set is due to Iwata [97].

From the graph-theoretical point of view, the graph invariant $\gamma(G)$ — the size of the minimum dominating set in $G$ — is also intensively studied along with its many variants. For example, the survey of Hedetniemi and Laskar from 1990 [92] lists over 300 papers on domination in graphs. If we consider the broader family of all covering problems, the list becomes even larger [90].

Let us mention here two examples of directions of research in the theory of graph domination. It is easy to see that a connected graph $G$ admits a dominating set with at most $|V(G)|/2$ vertices: we simply take into the dominating set every second layer of a spanning tree of $G$. A deep result by Reed [119] improves this bound to $3|V(G)|/8$ for graphs of minimum degree at least three. It is conjectured that, maybe with some other mild assumptions, such graphs should admit a dominating set of size $|V(G)|/3$.

A second example is a conjecture of Vizing from 1963 [128] that asserts that a domination number of a cartesian product of two graphs is not smaller than the product of the domination numbers of the ingredient graphs. Although this conjecture has been settled for many subcases [7, 37], it seems that we are still far away from obtaining a final answer [22].

In this and the next section we study the time complexity of determining the values of two variants of the domination number $\gamma(G)$ — namely, the Capacitated Dominating Set and the Minimum Maximal Irredundant Set problems.

Let $G = (V, E)$ be an undirected graph with $n$ vertices. In the Capacitated Dominating Set problem each vertex $v$ is additionally equipped with a number $c(v)$, called the capacity of $v$, which is the number of other vertices this vertex can dominate. Formally, we say that a set $S \subseteq V$ is a

capacitated dominating set if there exists $f_S : V \setminus S \to S$ such that $f_S(v)$ is a neighbour of $v$ for each $v \in V \setminus S$ and $|f_S^{-1}(v)| \leq c(v)$ for each $v \in S$. The function $f_S$ is called a *dominating function* for the set $S$. The Capacitated Dominating Set problem asks for the smallest possible size of a capacitated dominating set.

| Capacitated Dominating Set |
| --- |
| **Input:** A graph $G = (V, E)$ together with a function $c : V \to \mathbb{N}$. |
| **Task:** Find a smallest possible capacitated dominating set $S \subseteq V$. |

Note that if we take $c(v)$ to be the degree of $v$, we obtain the classical Dominating Set problem. In the case of the Dominating Set problem, checking whether a set $S \subseteq V$ is a dominating set in $G$ can be done in linear time directly from the definition. In the case of the Capacitated Dominating Set problem, this still can be done a polynomial time, but we need max–flow or maximum matching techniques.

The problem of solving Capacitated Dominating Set faster than the check-all-subsets $O^\star(2^n)$ time algorithm was posted by van Rooij in 2008 at Dagstuhl seminar [74] and on IWPEC 2008 open problem list. At first glance breaking $O^\star(2^n)$ barrier for the Capacitated Dominating Set problem seems a hard task, since the backtracking approach with a Measure&Conquer analysis [72] for the Dominating Set problem does not seem to extend to the capacitated variant. This is caused by the fact that the backtracking approach heavily relies on some local greedy decisions that can be performed in the Dominating Set case, but are significantly harder in the capacitated version due to the existence of a dominating function $f_S$. Even if the set $S$ is given, to compute a dominating function $f_S$ we need to use matching or max–flow techniques, and a minor change to $f_S$ in one part of the graph can influence the behaviour of $f_S$ in the whole graph. Thus, the backtracking approach seems of little use in the Capacitated Dominating Set problem.

We provide an algorithm which solves the Capacitated Dominating Set problem in $O(1.89^n)$ and polynomial space. The algorithm constructs $O^\star(\binom{n}{n/3}) = O(1.89^n)$ reductions of the input graph into a Constrained Capacitated Dominating Set problem instance (defined in Section 2.2.1), each solvable in polynomial time.

Our algorithm for Capacitated Dominating Set is somewhat similar to one of the first algorithms to break $O^\star(2^n)$ for the classical Dominating Set problem, namely the algorithm of Randerath and Schiermeyer [122].

Their algorithm also involves matching arguments and our algorithm, applied to DOMINATING SET, can be viewed as a simplification of their algorithm. However we do not know whether their algorithm could be used to solve the CAPACITATED DOMINATING SET problem.

### 2.2.1 CONSTRAINED CAPACITATED DOMINATING SET

In this section we introduce a constrained version of the CAPACITATED DOMINATING SET problem, namely the CONSTRAINED CAPACITATED DOMINATING SET problem, which can be solved in polynomial time.

The input of CONSTRAINED CAPACITATED DOMINATING SET is an undirected graph $G = (V, E)$, a set $U \subseteq V$ and a capacity function $c : V \to \{0, \ldots, n-1\}$. We ask for a smallest capacitated dominating set $S \subseteq V$ containing $U$ such that each vertex outside $U$ dominates at most one other vertex. Formally we ask for a dominating function $f_S$ satisfying

$$\begin{aligned} |f_S^{-1}(v)| &\leq 1 \quad \text{for each} \quad v \in S \setminus U \\ |f_S^{-1}(v)| &\leq c(v) \quad \text{for each} \quad v \in U \end{aligned} \tag{2.1}$$

Let $G = (V, E)$ with $U \subseteq V$ and a capacity function $c : V \to \{0, \ldots, n-1\}$ be a CONSTRAINED CAPACITATED DOMINATING SET instance. Consider a new graph $G' = (V', E')$ which can be constructed as follows. We begin with $V'$ and $E'$ empty.

- for any $v \in V \setminus U$ add $v$ to $V'$;
- for any $v \in U$ add $c(v)$ copies $v_1, v_2, \ldots, v_{c(v)}$ of $v$ to $V'$;
- for any $v \in V \setminus U$ and $u \in U$ add an edge $u_i v$ to $E'$ for all $i$ iff $uv \in E$;
- for any $v, w \in V \setminus U$ add $vw$ to $E'$ iff $vw \in E$ and $c(v) + c(w) > 0$.

Note that there are no edges of the form $v_i w_j$ or $v_i v_j$ for $v, w \in U$.

We show a correspondence between feasible solutions of CONSTRAINED CAPACITATED DOMINATING SET in $G$ and matchings in $G'$.

**Lemma 2.2.** *Let $(G, c, U)$ be an instance of the* CONSTRAINED CAPACITATED DOMINATING SET *problem. For any feasible solution $S \subseteq V$ with a dominating function $f_S$ satisfying the condition* (2.1)*, one may construct in polynomial time a matching $\phi(S, f_S)$ in $G'$ satisfying $|\phi(S, f_S)| = |V| - |S|$.*

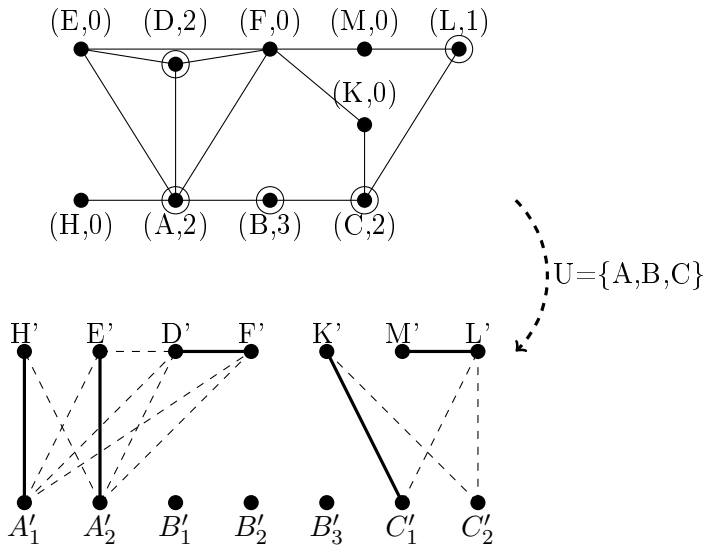


Figure 2.1: From the constrained capacitated dominating set $\{A, B, C, D, L\}$ (where $U = \{A, B, C\}$) to a matching. By a pair $(X, i)$ we denote a vertex $X$ with its capacity $c(X) = i$.

*Proof.* Let us define the matching $\phi(S, f_S)$ as follows (see Figure 2.1 for an illustration):

- for each $v \notin S$ such that $f_S(v) \notin U$ add $vf_S(v)$ to $\phi(S, f_S)$;
- for each $v \notin S$ such that $u = f_S(v) \in U$ add $vu_i$ to $\phi(S, f_S)$, where $u_i$ is a copy of $u$ in $G'$ and different copies $u_i$ are chosen for different vertices $v$ with $f_S(v) = u$ (note that $|f_S^{-1}(u)| \leq c(u)$, so there are enough vertices $u_i$).

Note that every vertex $v \in V \setminus S$ is an endvertex of an edge in the matching $\phi(S, f_S)$. The second endvertex is $f_S(v)$ (in the case $f_S(v) \notin U$) or a copy of $f_S(v)$ in $G'$ (in the case $f_S(v) \in U$). Note that by condition 2.1, every vertex $w \in S \setminus U$ is an endvertex of at most one chosen edge, thus $\phi(S, f_S)$ is indeed a matching. Moreover, every edge in $\phi(S, f_S)$ has exactly one endpoint in $V \setminus S$. Therefore $|\phi(S, f_S)| = |V| - |S|$.

□

**Lemma 2.3.** *For any matching $M$ in the graph $G'$, one may construct in polynomial time a feasible solution $\psi(M)$ to the* Constrained Capacitated Dominating Set *instance $(G, c, U)$ with dominating function $f_{\psi(M)}$ satisfying $|\psi(M)| = |V| - |M|$.*

*Proof.* Consider a capacitated dominating set $\psi(M)$ with dominating function $f_{\psi(M)}$, constructed as follows. We start with $\psi(M)$ and $f_{\psi(M)}$ empty.

- add all vertices of $U$ to $\psi(M)$;
- for $u \in U$ and for each $i$ such that $u_i v \in M$, set $f_{\psi(M)}(v) = u$;
- for any edge $vw \in M$, where $v, w \notin U$ one of the endpoints (say $v$) has to satisfy $c(v) > 0$, add $v$ to $\psi(M)$ and set $f_{\psi(M)}(w) = v$;
- for any $v \notin U$ which is not an endpoint of any edge in $M$, add $v$ to $\psi(M)$.

It is easy to verify that the above procedure does indeed give a feasible solution to Constrained Capacitated Dominating Set. We have $|\psi(M)| = |V| - |M|$ since for each edge in $M$, exactly one of its endpoints does not belong to $\psi(M)$. □

We conclude this section with the following theorem.

**Theorem 2.4.** *The* Constrained Capacitated Dominating Set *problem can be solved in polynomial time.*

*Proof.* By Lemmata 2.2 and 2.3, in order to find the solution of the Constrained Capacitated Dominating Set problem it is enough to find any maximum matching in $G'$, which can be done in polynomial time (see e.g. [113]). □

### 2.2.2 From Constrained Capacitated Dominating Set to Capacitated Dominating Set

Let us start with the following simple observation. Let $S$ be any capacitated dominating set and let $f_S$ be a dominating function for $S$. Let

$$U_S = \{v \in S : |f_S^{-1}(v)| \geq 2\}.$$

Since every vertex is dominated by exactly one vertex, and each vertex in $U_s$ dominates at least 3 vertices (including itself), so $|U_S| \leq n/3$. Moreover, $S$ with the function $f_S$ is a feasible solution for the Constrained Capacitated Dominating Set instance with the graph $G$ and the set $U_S$. Therefore the following algorithm solves Capacitated Dominating Set:

1. For each $U \subseteq V$ satisfying $|U| \leq n/3$ solve the Constrained Capacitated Dominating Set instance with graph $G$ and subset $U$.

2. Return the smallest capacitated dominating set from the constructed Constrained Capacitated Dominating Set instances.

The Constrained Capacitated Dominating Set problem can be solved in polynomial time and there are

$$\sum_{k=0}^{\lceil n/3 \rceil} \binom{n}{k} = O^\star\left(\binom{n}{\lceil n/3 \rceil}\right).$$

possible sets $U$ (i.e. sets of cardinality at most $n/3$). By Lemma 2.1

$$O^\star\left(\binom{n}{\lceil n/3 \rceil}\right) = O^\star\left(\left(\frac{1}{\left(\frac{1}{3}\right)^{\frac{1}{3}}\left(\frac{2}{3}\right)^{\frac{2}{3}}}\right)^n\right) = O(1.8899^n),$$

thus the whole algorithm works in $O(1.89^n)$ time.

## 2.3 Irredundant set

In this section we study a second variant of the dominating set, namely the irredundant set. We say a set $S \subseteq V$ is *irredundant* if for any $v \in S$ there exists a vertex $u \in V$ such that $v$ dominates $u$ and $S \setminus \{v\}$ does not dominate $u$. We call any such vertex $u$ a *private* vertex for $v$. An irredundant set is called (inclusion-wise) *maximal* if it is not a proper subset of any other irredundant set. Note that an maximal irredundant set does not necessarily have to dominate the whole vertex set of $G$ as in Figure 2.2.

The upper irredundance number $IR(G)$ is defined as the largest possible cardinality of an irredundant set in $G$, whereas the lower irredundance number $ir(G)$ is defined as the smallest possible cardinality of an (inclusion-wise)

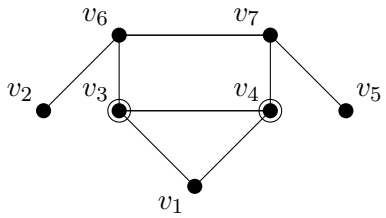


Figure 2.2: The smallest maximal irredundant set $X = \{v_3, v_4\}$. Note that $X$ is not a dominating set.

maximal irredundant set. The problems Minimum Maximal Irredundant Set and Maximum Irredundant Set ask for $ir(G)$ and $IR(G)$ respectively.

| Minimum Maximal Irredundant Set |
|---|
| **Input:** An undirected graph $G$. |
| **Task:** Compute $ir(G)$. |

| Maximum Irredundant Set |
|---|
| **Input:** An undirected graph $G$. |
| **Task:** Compute $IR(G)$. |

The irredundance numbers are interesting from the graph-theoretic point of view due to their relation to numerous other graph parameters. These relations has been established in around 100 research papers [66], e.g., [3, 19, 20, 28, 38, 40, 64, 65, 68, 91, 104].

Let us make an example of the aforementioned relations. It is easy to relate $ir(G)$ to the dominating number $\gamma(G)$.

**Proposition 2.5** ([39])**.** *Any inclusion-wise minimal dominating set in a graph $G$ is an inclusion-wise maximal irredundant set in $G$, too.*

*Proof.* Let $S \subseteq V(G)$ be an inclusion-wise minimal dominating set in $G$. As $S$ is inclusion-wise minimal, for any $v \in S$ the set $S \setminus \{v\}$ does not dominate $V(G)$, thus there exists $u_v \in V(G)$ that is dominated by $v$, but not by $S \setminus \{v\}$. We infer that $S$ is an irredundant set in $G$. To see that $S$ is inclusion-wise maximal, note that for any $w \in V(G) \setminus S$ the set $S' = S \cup \{w\}$ is not irredundant, as $S = S' \setminus \{w\}$ dominates $V(G)$ and thus $w$ cannot have a private vertex. □

We infer that $ir(G) \leq \gamma(G)$. Let us recall that a set $I \subseteq V(G)$ is *independent*, if $G[I]$ does not contain any edge. The graph invariant $\alpha(G)$ is defined as the cardinality of the largest independent set in $G$. Note the following simple fact.

**Proposition 2.6.** *Let $I$ be an independent set in $G$. Then $I$ is an irredundant set. Moreover, if $I$ is inclusion-wise maximal, then $I$ is a dominating set in $G$.*

*Proof.* To see that $I$ is an irredundant set, note that for any $v \in I$, $v$ can serve as a private vertex for $v$. If $I$ is not a dominating set in $G$, there exists $u \in V(G) \setminus N_G[I]$, $I \cup \{u\}$ is an independent set, and $I$ is not inclusion-wise maximal. □

By the above reasonings we obtained the well-known *domination chain*:

$$ir(G) \leq \gamma(G) \leq \alpha(G) \leq IR(G).$$

Moreover, let us mention that it is known that $ir(G)$ is always within a constant factor of $\gamma(G)$ [3, 19]:

$$\gamma(G)/2 < ir(G) \leq \gamma(G) \leq 2 \cdot ir(G) - 1.$$

Verifying whether a set $S \subseteq V(G)$ is an irredundant set in a graph $G$ can be done in linear time directly from the definition. Thus, both $ir(G)$ and $IR(G)$ can be computed in $O^\star(2^n)$ time by an exhaustive enumeration. The problems of solving the Minimum Maximal Irredundant Set and Maximum Irredundant Set problems faster than this obvious $O^\star(2^n)$ algorithms were posed by van Rooij in 2008 [74]. In this section we present an algorithm that solves Minimum Maximal Irredundant Set in $O(1.999956^n)$time. In [48] we gave an $O(1.9657^n)$ algorithm for Maximum Irredundant Set, but, since it is a quite straightforward branching algorithm with a bit tedious analysis, we do not include it in this dissertation.

Our algorithm for Minimum Maximal Irredundant Set is a simple iterative backtracking algorithm working in polynomial space. The interesting part is its analysis, where we prove some structural properties of the considered problem. We would like to note that our techniques seem a bit similar to those that lead to $O^\star((2-\varepsilon)^n)$-time algorithms for Domatic Number and TSP in graphs of bounded degree [12]. We think it is interesting and somewhat surprising that such techniques can be used in graphs without any degree assumption.

We start with a simple observation on irredundant sets.

**Proposition 2.7.** *Let $G$ be a graph, $S \subseteq V(G)$ be an irredundant set, and $v \in V(G)$ be a vertex in $G$. If $N[v] \subseteq S$, then $v$ is an isolated vertex in $G$ (i.e., $N_G(v) = \emptyset$).*

*Proof.* If $v$ is not an isolated vertex in $G$, then $N_G(v) \subseteq S \setminus \{v\}$ dominates $N_G[v]$ and $v$ cannot have a private vertex. □

Let us now proceed to the description of the algorithm. W.l.o.g. we may assume that $G$ contains no isolated vertices, since they need to be included in any maximal irredundant set.

Let $\mathcal{F}_k$ be the family of irredundant sets in $G$ of size not greater than $k$. Note that checking if a set is a (maximal) irredundant set can be done in polynomial time. Moreover, the family of irredundant sets is closed under taking subsets. Therefore $\mathcal{F}_k$ can be enumerated in $O^\star(|\mathcal{F}_k|)$ time and polynomial space by a simple backtracking algorithm EnumerateIrredundantSets$(G, k)$.

```
Function IrredundantSearch(G, k, S, Ŝ)
 1: if S is an irredundant set in G then
 2:     output S
 3:     if |S| < k then
 4:         for all v ∈ V(G) \ (S ∪ Ŝ) do
 5:             IrredundantSearch(G, k, S ∪ {v}, Ŝ)
 6:             Ŝ := Ŝ ∪ {v}
Function EnumerateIrredundantSets(G, k)
 7: IrredundantSearch(G, k, ∅, ∅)
```

To see that EnumerateIrredundantSets$(G, k)$ works in $O^\star(|\mathcal{F}_k|)$ time note that each call to the IrredundantSearch procedure takes polynomial time and outputs a new irredundant set in $G$. The set $\hat{S}$ should be interpreted as the set of vertices that were chosen not to be included in the currently constructed irredundant set.

Consider a simple iterative backtracking algorithm that enumerates $\mathcal{F}_k$ for $k = 0, 1, 2, \ldots, n$ until it finds an (inclusion-wise) maximal irredundant set. Now we prove that it works in $O(1.999956^n)$ time.

First, as a warm-up, let us show a $O^\star((2 - \varepsilon_\Delta)^n)$ time bound for graphs with maximum degree bounded by $\Delta$, where $\varepsilon_\Delta$ depends on $\Delta$. Construct a set $A \subseteq V$ as follows: repeatedly add any vertex $v \in V$ to $A$ and remove from $V$ all vertices distant by at most 2 from $v$. At each step, at most $1 + \Delta + \Delta(\Delta - 1) = 1 + \Delta^2$ vertices are removed, therefore $|A| \geq n/(1 + \Delta^2)$.

The set $A$ is an independent set; moreover, closed neighbourhoods $\{N[v] : v \in A\}$ are disjoint. If $S$ in an irredundant set, then $N[v] \not\subseteq S$ (Proposition 2.7) and, therefore, for each $v \in A$ we have at most $2^{|N[v]|} - 1$ possibilities to choose $S \cap N[v]$ instead of $2^{|N[v]|}$. As these neighbourhoods are disjoint, this leads to the following bound:

$$|\mathcal{F}_n| \leq 2^n \prod_{v \in A} \frac{2^{|N[v]|} - 1}{2^{|N[v]|}} \leq 2^n \left( \frac{2^{\Delta+1} - 1}{2^{\Delta+1}} \right)^{\frac{n}{1+\Delta^2}} = (2 - \varepsilon_\Delta)^n,$$

and the time bound for the algorithm follows.

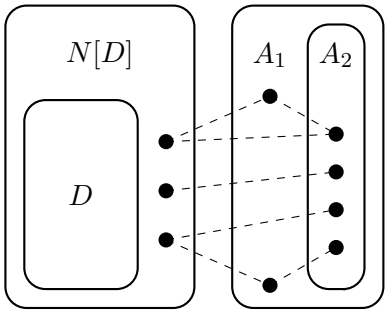


Figure 2.3: Notation in the proof of Lemma 2.8

Now we show how to bypass the maximum degree assumption. By Proposition 2.5, if $G$ admits a dominating set of size not greater than $149n/300$, then it admits an inclusion-wise maximal irredundant set of the same size and the algorithm stops before or at the step $k = \lfloor 149n/300 \rfloor$ and up to this point consumes $O^\star(\binom{n}{149n/300}) = O(1.999956^n)$ time (Lemma 2.1). Therefore we may consider only the case where every dominating set in $G$ is of size greater than $149n/300$.

The following structural lemma is crucial for the analysis.

**Lemma 2.8.** *Let $G = (V, E)$ be a graph with $n$ vertices that contains no dominating set of cardinality smaller than $149n/300$. Then there exists a set $A \subseteq V$ satisfying:*

1. *$A$ is an independent set and the neighbourhoods $\{N[v] : v \in A\}$ are disjoint,*
2. *every vertex in $A$ has degree at most 6,*

*3.* $|A| \geq 41n/9800$.

*Proof.* We construct a dominating set $D$ greedily. Start with $D = \emptyset$. In a single step, take any vertex $v$ that adds at least 3 new vertices to $N[D]$, i.e. $|N[D \cup \{v\}] \setminus N[D]| \geq 3$, and add $v$ to $D$. This algorithm stops at some point and let $A_1 = V \setminus N[D]$, i.e. the vertices not dominated by $D$. For every vertex $v$ we have $|N[v] \cap A_1| \leq 2$, since $D$ cannot be extended any more. In particular, every vertex in $G[A_1]$ has degree at most 1, so $G[A_1]$ is a graph of isolated vertices and isolated edges. Let $A_2$ be any maximal independent set in $G[A_1]$, i.e. $A_2$ contains all isolated vertices of $G[A_1]$ and one endpoint of every isolated edge (see Figure 2.3). The set $A_2$ is an independent set in $G$, too.

Let us now note that $D \cup A_2$ is a dominating set in $G$, since $A_2$ dominates $A_1$. Therefore $|D| + |A_2| \geq 149n/300$. Note that, by the construction procedure of $D$, we have $|D| \leq \frac{1}{3}|N[D]| = \frac{1}{3}|V \setminus A_1|$, so:

$$149/300 \leq \frac{|D| + |A_2|}{|V|} \leq \frac{1}{3} - \frac{|A_1|}{3|V|} + \frac{|A_2|}{|V|} \leq \frac{1}{3} + \frac{2}{3} \cdot \frac{|A_2|}{|V|}.$$

Therefore $|A_2| \geq 49n/200$.

Now recall that every vertex in $V$ has at most two vertices from $A_1$ in its closed neighbourhood. Therefore, every vertex in $V$ has at most two neighbours in $A_2$. Let $n_7$ be the number of vertices in $A_2$ with degree at least 7. By counting edge endpoints we obtain that $7n_7 \leq 2(n - |A_2|) \leq 151n/100$ and $n_7 \leq 151n/700$. Let $A_3 \subseteq A_2$ be the set of vertices of degree at most 6. Then $|A_3| \geq 41n/1400$.

Now construct $A \subseteq A_3$ greedily. In a single step, add any $v \in A_3$ to $A$ and remove from $A_3$ the vertex $v$ and all vertices that share a neighbour with $v$ (recall that $A_3$ is an independent set). Since the vertices in $A_3$ have degree at most 6 and every vertex in $V$ is a neighbour of at most two vertices in $A_3$, then at one step we remove at most 7 vertices from $A_3$. Therefore $|A| \geq 41n/9800$. □

The bound for our iterative-DFS algorithm is now straightforward. Note that for every non–isolated vertex $v$ at least one point from $N[v]$ does not belong to an irredundant set (Proposition 2.7). By Lemma 2.8 we obtain $41n/9800$ disjoint sets $\{N[v] : v \in A\}$, such that all these sets are of size at most 7 and no $N[v]$ can be contained in an irredundant set. Therefore the

total number of irredundant sets is bounded by:

$$2^n \cdot \left(\frac{2^7-1}{2^7}\right)^{\frac{41n}{9800}} = O(1.99994^n).$$

## 2.4 Scheduling jobs with precedences

The area of scheduling algorithms originates from practical questions regarding scheduling jobs on single- or multiple-processor machines or scheduling I/O requests. It has quickly become one of the most important areas in algorithmics, with significant influence on other branches of computer science. For example, the research of the job-shop scheduling problem in 1960s resulted in designing the competitive analysis [84], initiating the research of online algorithms. Up to today, the scheduling literature consists of thousands of research papers. We refer the reader to the classical textbook of Brucker [23].

Among scheduling problems one may find a bunch of problems solvable in polynomial time, as well as many NP-hard ones. For example, the aforementioned job-shop problem is NP-complete on at least three machines [106], but polynomial on two machines with unitary processing times [93].

Scheduling problems come in numerous variants. For example, one may consider scheduling on one machine, or many uniform or non-uniform machines. The jobs can have different attributes: they may arrive at different times, may have deadlines or precedence constraints, preemption may or may not be allowed. There are also many objective functions, for example the makespan of the computation, total completion time, total lateness (in case of deadlines for jobs) etc.

Let us focus on the case of a single machine. Assume we are given a set of jobs $V$, and each job $v$ has its processing time $t(v) \in (0, +\infty)$. For a job $v$, its completion time is the total amount of time that this job waited to be finished; formally, the completion time of a job $v$ is defined as a sum of processing times of $v$ and all jobs scheduled earlier. If we are to minimize the total completion time (i.e, the sum of completion times over all jobs), it is clear that the jobs should be scheduling in order of increasing processing times. The question of minimizing the makespan of the computation (i.e., maximum completion time) is obvious in this setting, but we note that minimizing makespan is polynomially solvable even if we are given a precedence constraints on the jobs (i.e., a partial order on the set of jobs is given, and

a job cannot be scheduled before all its predecessors in the partial order are finished) and jobs arrive at different times (i.e., each job has its arrival time, before which it cannot be scheduled) [105].

Lenstra and Rinnooy Kan [107] in 1978 proved that the question of minimizing total completion time on one machine becomes NP-complete if we are given precedence constraints on the set of jobs. To the best of our knowledge the currently smallest approximation ratio for this case equals 2, due to independently discovered algorithms by Chekuri and Motwani [30] as well as Margot et al. [109]. The problem of minimizing total completion time on one machine, given precedence constraints on the set of jobs can be solved by a standard dynamic programming algorithm in time $O^\star(2^n)$, where $n$ denotes the number of jobs. The goal of this section is to break the $2^n$-barrier for this problem.

Before we start, let us define formally the considered problem. As in this section we focus on a single scheduling problem, for brevity we denote it by SCHED.

SCHED
**Input:** A partially ordered set $(V, \leq)$, (the elements of which are called *jobs*) together with a function $t : V \to (0, +\infty)$ (for a job $v \in V$ the value $t(v)$ is called a *processing time* of $v$).
**Task:** Compute a bijection $\sigma : V \to \{1, 2, \ldots, |V|\}$ (called an *ordering*) that satisfies the precedence constraints (i.e., if $u < v$, then $\sigma(u) < \sigma(v)$) and minimizes the total completion time of all jobs defined as

$$T(\sigma) = \sum_{v \in V} \sum_{u:\sigma(u) \leq \sigma(v)} t(u) = \sum_{v \in V} (|V| - \sigma(v) + 1) t(v).$$

If $u < v$ for $u, v \in V$ (i.e., $u \leq v$ and $u \neq v$), we say that $u$ *precedes* $v$, $u$ is a *predecessor* or *prerequisite* of $v$, $u$ is *required* for $v$ or that $v$ is a *successor* of $v$. We denote $|V|$ by $n$.

As discussed earlier, scheduling problems come in numerous variants and, to keep track of all possibilities, Graham, Lawler, Lenstra and Rinnooy Kan introduced the so-called Graham notation. In this notation each problem's name consists of three parts: $\alpha|\beta|\gamma$; $\alpha$ describes assumptions on machines (e.g., one machine or uniform machines), $\beta$ describes assumptions on jobs (e.g., preemption allowed, precedences) and $\gamma$ describes the objective function (e.g., minimize makespan or total completion time). In this notation, the

SCHED problem is called $1|\text{prec}|\sum C_i$. Here, 1 stands for one machine, prec indicates that we are given precedence constraints on the set of jobs, and $\sum C_i$ denotes that we are to minimize total completion time (the sum of completion times $C_i$).

SCHED is a special case of the precedence constrained Travelling Repairman Problem (prec-TRP), defined as follows. A repairman needs to visit all vertices of a (directed or undirected) graph $G = (V, E)$ with distances $d : E \to [0, \infty)$ on edges. At each vertex, the repairman is supposed to repair a broken machine; a cost of a machine $v$ is the time $C_v$ that it waited before being repaired. Thus, the goal is to minimize the total repair time, that is, $\sum_{v \in V} C_v$. Additionally, in the precedence constrained case, we are given a partial order $(V, \leq)$ on the set of vertices of $G$; a machine can be repaired only if all its predecessors are already repaired. Note that, given an instance $(V, \leq, t)$ of SCHED, we may construct equivalent prec-TRP instance, by taking $G$ to be a complete directed graph on the vertex set $V$, keeping the precedence constraints unmodified, and setting $d(u, v) = t(v)$.

The TRP problem is closely related to the Traveling Salesman Problem (TSP). All these problems are NP-complete and solvable in $O^\star(2^n)$ time by an easy application of the dynamic programming approach (here $n$ stands for the number of vertices in the input graph). In 2009, Björklund [10] discovered a genuine way to solve probably the easiest NP-complete version of the TSP problem — the question of deciding whether the input undirected graph is Hamiltonian — in randomized $O(1.66^n)$ time. However, his approach does not extend to directed graphs, not even mentioning graphs with distances defined on edges.

Björklund's approach is based on purely graph-theoretical and combinatorial reasonings, and seem unable to cope with arbitrary (large, real) weights (distances, processing times). This is also the case with many other combinatorial approaches. Probably motivated by this, Woeginger at International Workshop on Parameterized and Exact Computation (IWPEC) in 2004 [129] has posed the question (repeated in 2008 [130]), whether it is possible to construct an $O((2 - \varepsilon)^n)$ time algorithm for at the SCHED problem[1]. This problem seems to be the easiest case of the aforementioned family of TSP-related problems with arbitrary weights. In this section we present such an algorithm, thus affirmatively answering Woeginger's ques-

[1]Although Woeginger in his papers asks for an $O(1.99^n)$ algorithm, the intention is clearly to ask for an $O((2 - \varepsilon)^n)$ algorithm.

tion. Woeginger also asked [129, 130] whether an $O((2-\varepsilon)^n)$ time algorithm for one of the problems TRP, TSP, prec-TRP, SCHED implies $O((2-\varepsilon)^n)$ time algorithms for the other problems. This problem is still open.

The most important ingredient of our algorithm is a combinatorial lemma (Lemma 2.13) which allows us to investigate the structure of the SCHED problem. We heavily use the fact that we are solving the SCHED problem and not its more general TSP related version, and for this reason we believe that obtaining $O(2-\varepsilon)^n$ time algorithms for other problems listed by Woeginger is much harder.

### 2.4.1 High-level overview — part 1

Let us recall that our task in the SCHED problem is to compute an ordering $\sigma : V \to \{1, 2, \ldots, n\}$ that satisfies the precedence constraints (i.e., if $u < v$ then $\sigma(u) < \sigma(v)$) and minimizes the total completion time of all jobs defined as

$$T(\sigma) = \sum_{v \in V} \sum_{u:\sigma(u) \leq \sigma(v)} t(u) = \sum_{v \in V} (n - \sigma(v) + 1)t(v).$$

We define *the cost of job $v$ at position $i$* to be $T(v, i) = (n-i+1)t(v)$. Thus, the total completion time is the total cost of all jobs at their respective positions in the ordering $\sigma$.

We begin by describing the algorithm that solves SCHED in $O^\star(2^n)$ time, which we call *the DP algorithm* — this will be the basis for our further work. The idea — a standard dynamic programming over subsets — is that if we decide that a particular set $X \subseteq V$ will (in some order) form the prefix of our optimal $\sigma$, then the order in which we take the elements of $X$ does not affect the choices we make regarding the ordering of the remaining $V \setminus X$; the only thing that matters are the precedence constraints imposed by $X$ on $V \setminus X$. Thus, for each candidate set $X \subseteq V$ to form a prefix, the algorithm computes a bijection $\sigma[X] : X \to \{1, 2, \ldots, |X|\}$ that minimizes the cost of jobs from $X$, i.e., it minimizes $T(\sigma[X]) = \sum_{v \in X} T(v, \sigma[X](v))$. The value of $T(\sigma[X])$ is computed using the following easy to check recursive formula:

$$T(\sigma[X]) = \min_{v \in \max(X)} \left[ T(\sigma[X \setminus \{v\}]) + T(v, |X|) \right]. \tag{2.2}$$

Here, by $\max(X)$ we mean the set of maximum elements of $X$ — those which do not precede any element of $X$. The bijection $\sigma[X]$ is constructed by prolonging $\sigma[X \setminus \{v\}]$ by $v$, where $v$ is the job at which the minimum is

attained. Notice that $\sigma[V]$ is exactly the ordering we are looking for. We calculate $\sigma[V]$ recursively, using formula (2.2), storing all computed values $\sigma[X]$ in memory to avoid recomputation. Thus, as the computation of a single $\sigma[X]$ value given all the smaller values takes polynomial time, while $\sigma[X]$ for each $X$ is computed at most once the whole algorithm indeed runs in $O^\star(2^n)$ time.

The overall idea of our algorithm is to identify a family of sets $X \subseteq V$ that — for some reason — are not reasonable prefix candidates, and we can skip them in the computations of the DP algorithm; we will call these *unfeasible sets*. If the number of feasible sets is not larger than $c^n$ for some $c < 2$, we will be done — our recursion will visit only feasible sets, assuming $T(\sigma[X])$ to be $\infty$ for unfeasible $X$ in formula (2.2), and the running time will be $O^\star(c^n)$. This is formalized in the following proposition.

**Proposition 2.9.** *Assume we are given a polynomial-time algorithm $\mathcal{R}$ that, given a set $X \subseteq V$, either accepts it or rejects it. Moreover, assume that the number of sets accepted by $\mathcal{R}$ is bounded by $O(c^n)$ for some constant $c$. Then one can find in time $O^\star(c^n)$ an optimal ordering of the jobs in $V$ among those orderings $\sigma$ where $\sigma^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$ for all $1 \leq i \leq n$, whenever such ordering exists.*

*Proof.* As discussed before, calculate $\sigma[V]$ recursively, using formula (2.2), storing all computed values $\sigma[X]$ in memory to avoid recomputation. Whenever we access a value $T(\sigma[X])$ for a set $X$ not accepted by $\mathcal{R}$, we take $T(\sigma[X]) = \infty$. As each application of the formula (2.2) gives at most $n$ recursive calls, the bound follows. □

### 2.4.2 The large matching case

We begin by noticing that the DP algorithm needs to compute $\sigma[X]$ only for those $X \subseteq V$ that are downward closed, i.e., if $v \in X$ and $u < v$ then $u \in X$. If there are many constraints in our problem, this alone will suffice to limit the number of feasible sets considerably, as follows. Construct an undirected graph $G$ with the vertex set $V$ and edge set $E = \{uv : u < v \vee v < u\}$. Let $M$ be a maximum matching[2] in $G$, which can be found in polynomial time [113]. If $X \subseteq V$ is downward closed, and $uv \in M$, $u < v$, then it is not possible that $u \notin X$ and $v \in X$. Obviously checking if a subset is downward

[2]Even an inclusion-maximal matching, which can be found greedily, is enough.

closed can be performed in polynomial time, thus we can apply Proposition 2.9, accepting only downward closed subsets of $V$. This leads to the following lemma:

**Lemma 2.10.** *The number of downward closed subsets of $V$ is bounded by $2^{n-2|M|}3^{|M|}$. If $|M| \geq \varepsilon_1 n$, then we can solve the* SCHED *problem in time*

$$T_1(n) = O^\star((3/4)^{\varepsilon_1 n} 2^n).$$

□

Note that for any small positive constant $\varepsilon_1$ the complexity $T_1(n)$ is of required order, i.e., $T_1(n) = O(c^n)$ for some $c < 2$ that depends on $\varepsilon_1$. Thus, we only have to deal with the case where $|M| < \varepsilon_1 n$.

Let us fix a maximum matching $M$, let $W_1 \subseteq V$ be the set of endpoints of $M$, and let $I_1 = V \setminus W_1$. Note that, as $M$ is a maximum matching in $G$, no two jobs in $I_1$ are bound by a precedence constraint, and $|W_1| \leq 2\varepsilon_1 n$, $|I_1| \geq (1 - 2\varepsilon_1)n$.

### 2.4.3 High-level overview — part 2

We are left in the situation where there is a small number of "special" elements ($W_1$), and the bulk remainder ($I_1$), consisting of elements that are tied by precedence constraints only to $W_1$ and not to each other.

First notice that if $W_1$ was empty, the problem would be trivial: with no precedence constraints we should simply order the tasks from the shortest to the longest. Now let us consider what would happen if all the constraints between any $u \in I_1$ and $w \in W_1$ would be of the form $u < w$ — that is, if the jobs from $I_1$ had no predecessors. For any prefix set candidate $\tilde{X}$ we consider $X = \tilde{X} \cap I_1$. Now for any $x \in X$, $y \in I_1 \setminus X$ we have an alternative prefix candidate: the set $\tilde{X}' = (\tilde{X} \cup \{y\}) \setminus \{x\}$. If $t(y) < t(x)$, there has to be a reason why $\tilde{X}'$ is not a strictly better prefix candidate than $\tilde{X}$ — namely, there has to exist $w \in W_1$ such that $x < w$, but $y \not< w$.

A similar reasoning would hold even if not all of $I_1$ had no predecessors, but just some significant fraction $J$ of $I$ — again, the only feasible prefix candidates would be those in which for every $x \in X \cap J$ and $y \in J \setminus X$ there is a reason (either $t(x) < t(y)$ or an element $w \in W_1$ which requires $x$, but not $y$) not to exchange them. It turns out that if $|J| > \varepsilon_2 n$, where $\varepsilon_2 > 2\varepsilon_1$, this observation suffices to prove that the number of possible intersections of

feasible sets with $J$ is significantly smaller than $2^{|J|}$. This is formalized and proved in Lemma 2.13, and is the cornerstone of the whole result.

A typical application of this lemma is as follows: say we have a set $K \subseteq I_1$ of cardinality $|K| > 2j$, while we know for some reason that all the predecessors of elements of $K$ appear on positions $j$ and earlier. If $K$ is large (a constant fraction of $n$), this is enough to limit the number of feasible sets to $(2-\varepsilon)^n$. To this end it suffices to show that there are significantly less than $2^{|K|}$ possible intersections of a feasible set with $K$. Each such intersection consists of a set of at most $j$ elements (that will be put on positions 1 through $j$), and then a set in which every element has a reason not to be exchanged with something from outside the set — and there are relatively few of those by Lemma 2.13 — and when we do the calculations, it turns out the resulting number of possibilities is significantly smaller than $2^{|K|}$.

To apply this reasoning, we need to be able to tell that all the prerequisites of a given element appear at some position or earlier. To achieve this, we need to know the approximate positions of the elements in $W_1$. We achieve this by branching into $4^{|W_1|}$ cases, for each element $w \in W_1$ choosing to which of the four quarters of the set $\{1, \ldots, n\}$ will $\sigma_{opt}(w)$ belong. This incurs a multiplicative cost of $4^{|W_1|}$, which will be offset by the gains from applying Lemma 2.13.

We will now repeatedly apply Lemma 2.13 to obtain information about the positions of various elements of $I_1$. We will repeatedly say that if "many" elements (by which we always mean more than $\varepsilon n$ for some $\varepsilon$) do not satisfy something, we can bound the number of feasible sets, and thus finish the algorithm. For instance, look at those elements of $I_1$ which can appear in the first quarter, i.e., none of their prerequisites appear in quarters two, three and four. If there is significantly more than $n/2$ of them, we can apply the above reasoning for $j = n/4$ (Lemma 2.17). Subsequent lemmata bound the number of feasible sets if there are many elements that cannot appear in any of the two first quarters (Lemma 2.15), if significantly *less* than $n/2$ elements can appear in the first quarter (Lemma 2.17) and if a significant number of elements in the second quarter could actually appear in the first quarter (Lemma 2.18). We also apply similar reasoning to elements that can or cannot appear in the last quarter.

We end up in a situation where we have four groups of elements, each of size roughly $n/4$, split upon whether they can appear in the first quarter and whether they can appear in the last one; moreover, those that can appear in the first quarter will not appear in the second, and those that can appear in

the fourth will not appear in the third. This means that there are two pairs of parts which do not interact, as the set of places in which they can appear are disjoint. We use this independence of sorts to construct a different algorithm than the DP we used so far, which solves our problem in this specific case in time $O^\star(2^{3n/4+\varepsilon})$ (Lemma 2.19).

As can be gathered from this overview, there are many technical details we will have to navigate in the algorithm. This is made more precarious by the need to carefully select all the epsilons. We decided to use symbolic values for them in the main proof, describing their relationship appropriately, using four constants $\varepsilon_k$, $k = 1, 2, 3, 4$. The constants $\varepsilon_k$ are very small positive reals, and additionally $\varepsilon_k$ is significantly smaller than $\varepsilon_{k+1}$ for $k = 1, 2, 3$. At each step, we shortly discuss the existence of such constants. We discuss the choice of optimal values of these constants in Section 2.4.9, although the value we perceive in our algorithm lies rather in the existence of an $O^\star((2-\varepsilon)^n)$ algorithm than in the value of $\varepsilon$ (which is admittedly very small).

### 2.4.4 Technical preliminaries

We start with a few simplifications. First, we add a few dummy jobs with no precedence constraints and zero processing times, so that $n$ is divisible by four. Second, by slightly perturbing the jobs' processing times, we can assume that all processing times are pairwise different and, moreover, each ordering has different total completion time. This can be done, for instance, by replacing time $t(v)$ with a pair $(t(v), (n+1)^{\pi(v)-1})$, where $\pi : V \to \{1, 2, \ldots, n\}$ is an arbitrary numbering of $V$. The addition of pairs is performed coordinatewise, whereas comparison is performed lexicographically. Note that this in particular implies that the optimal solution is unique, we denote it by $\sigma_{opt}$. Third, at the cost of an $n^2$ multiplicative overhead, we guess the jobs $v_{begin} = \sigma_{opt}^{-1}(1)$ and $v_{end} = \sigma_{opt}^{-1}(n)$ and we add precedence constraints $v_{begin} < v < v_{end}$ for each $v \neq v_{begin}, v_{end}$. If $v_{begin}$ or $v_{end}$ were not in $W_1$ to begin with, we add them there.

A number of times our algorithm branches into several subcases, in each branch assuming some property of the optimal solution $\sigma_{opt}$. Formally speaking, in each branch we seek the optimal ordering among those that satisfy the assumed property. We somewhat abuse the notation and denote by $\sigma_{opt}$ the optimal solution in the currently considered subcase. Note that $\sigma_{opt}$ is always unique within any subcase, as each ordering has different total completion time.

For $v \in V$ by $pred(v)$ we denote the set $\{u \in V : u < v\}$ of predecessors of $v$, and by $succ(v)$ we denote the set $\{u \in V : v < u\}$ of successors of $v$. We extend this notation to subsets of $V$: $pred(U) = \bigcup_{v \in U} pred(v)$ and $succ(U) = \bigcup_{v \in U} succ(v)$. Note that for any set $U \subseteq I_1$, both $pred(U)$ and $succ(U)$ are subsets of $W_1$.

### 2.4.5 The core lemma

We now formalize the idea of exchanges presented in Section 2.4.3.

**Definition 2.11.** Consider some set $K \subseteq I_1$, and its subset $L \subseteq K$. If there exists $u \in L$ such that for every $w \in succ(u)$ we can find $v_w \in (K \cap pred(w)) \setminus L$ with $t(v_w) < t(u)$ then we say $L$ is *succ-exchangeable* with respect to $K$, otherwise we say $L$ is *non-succ-exchangeable* with respect to $K$.

Similarly, if there exists $v \in (K \setminus L)$ such that for every $w \in pred(v)$ we can find $u_w \in L \cap succ(w)$ with $t(u_w) > t(v)$, we call $L$ *pred-exchangeable* with respect to $K$, otherwise we call it *non-pred-exchangeable* with respect to $K$.

Whenever it is clear from the context, we omit the set $K$ with respect to which its subset is or is not pred- or succ-exchangeable.

The applicability of this definition is in the following observation:

**Observation 2.12.** *Let $K \subseteq I_1$. If for any $v \in K, w \in pred(K)$ we have that $\sigma_{opt}(v) > \sigma_{opt}(w)$, then for any $1 \leq i \leq n$ the set $K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is non-succ-exchangeable with respect to $K$.*

*Similarly, if for any $v \in K, w \in succ(K)$ we have $\sigma_{opt}(v) < \sigma_{opt}(w)$, then the sets $K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ are non-pred-exchangeable with respect to $K$.*

*Proof.* The proofs for the first and the second case are analogous. However, to help the reader get intuition on exchangeable sets, we provide them both in full detail. See Figure 2.4 for an illustration on the *succ*-exchangeable case.

Non-*succ*-exchangeable sets. Assume, by contradiction, that for some $i$ the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is *succ*-exchangeable. Let $u \in L$ be a job witnessing it. Let $w$ be the successor of $u$ with minimum $\sigma_{opt}(w)$ (there exists one, as $v_{end} \in succ(u)$). By Definition 2.11, we have $v_w \in (K \cap pred(w)) \setminus L$ with $t(v_w) < t(u)$. As $v_w \in K \setminus L$, we have $\sigma_{opt}(v_w) > \sigma_{opt}(u)$. As $v_w \in pred(w)$, we have $\sigma_{opt}(v_w) < \sigma_{opt}(w)$.

Consider an ordering $\sigma'$ defined as $\sigma'(u) = \sigma_{opt}(v_w)$, $\sigma'(v_w) = \sigma_{opt}(u)$ and $\sigma'(x) = \sigma_{opt}(x)$ if $x \notin \{u, v_w\}$; in other words, we swap the positions of $u$ and $v_w$ in the ordering $\sigma_{opt}$. We claim that $\sigma'$ satisfies all the precedence constraints. As $\sigma_{opt}(u) < \sigma_{opt}(v_w)$, $\sigma'$ may only violates constraints of the form $x < v_w$ and $u < y$. However, if $x < v_w$, then $x \in pred(K)$ and $\sigma'(v_w) = \sigma_{opt}(u) > \sigma_{opt}(x) = \sigma'(x)$ by the assumptions of the Observation. If $u < y$, then $\sigma'(y) = \sigma_{opt}(y) \geq \sigma_{opt}(w) > \sigma_{opt}(v_w) = \sigma'(u)$, by the choice of $w$. Thus $\sigma'$ is a feasible solution to the considered SCHED instance. Since $t(v_w) < t(u)$, we have $T(\sigma') < T(\sigma_{opt})$, a contradiction.

Non-*pred*-exchangeable sets. Assume, by contradiction, that for some $i$ the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is *pred*-exchangeable. Let $v \in (K \setminus L)$ be a job witnessing it. Let $w$ be the predecessor of $v$ with maximum $\sigma_{opt}(w)$ (there exists one, as $v_{begin} \in pred(v)$). By Definition 2.11, we have $u_w \in L \cap succ(w)$ with $t(u_w) > t(v)$. As $u_w \in L$, we have $\sigma_{opt}(u_w) < \sigma_{opt}(v)$. As $u_w \in succ(w)$, we have $\sigma_{opt}(u_w) > \sigma_{opt}(w)$.

Consider an ordering $\sigma'$ defined as $\sigma'(v) = \sigma_{opt}(u_w)$, $\sigma'(u_w) = \sigma_{opt}(v)$ and $\sigma'(x) = \sigma_{opt}(x)$ if $x \notin \{v, u_w\}$; in other words, we swap the positions of $v$ and $u_w$ in the ordering $\sigma_{opt}$. We claim that $\sigma'$ satisfies all the precedence constraints. As $\sigma_{opt}(u_w) < \sigma_{opt}(v)$, $\sigma'$ may only violates constraints of the form $x > u_w$ and $v > y$. However, if $x > u_w$, then $x \in succ(K)$ and $\sigma'(u_w) = \sigma_{opt}(v) < \sigma_{opt}(x) = \sigma'(x)$ by the assumptions of the Observation. If $v > y$, then $\sigma'(y) = \sigma_{opt}(y) \leq \sigma_{opt}(w) < \sigma_{opt}(u_w) = \sigma'(v)$, by the choice of $w$. Thus $\sigma'$ is a feasible solution to the considered SCHED instance. Since $t(u_w) > t(v)$, we have $T(\sigma') < T(\sigma_{opt})$, a contradiction. □

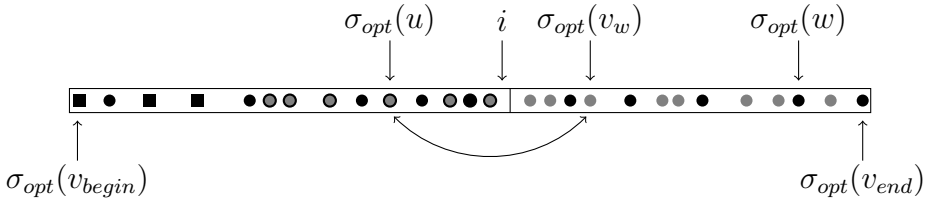


Figure 2.4: Figure illustrating the *succ*-exchangeable case of Observation 2.12. Gray circles indicate positions of elements of $K$, black contour indicates that an element is also in $L$. Black squares indicate positions of elements from $pred(K)$, and black circles — positions of other elements from $W_1$.

Observation 2.12 means that if we manage to identify a set $K$ satisfying the assumptions of the observation, the only sets the DP algorithm has to consider are the non-exchangeable ones. The following core lemma proves that there are few of those (provided that $K$ is big enough), and we can identify them easily.

**Lemma 2.13.** *For any set $K \subseteq I_1$ the number of non-succ-exchangeable (non-pred-exchangeable) subsets with regard to $K$ is at most $\sum_{l \leq |W_1|} \binom{|K|}{l}$. Moreover, there exists an algorithm which checks whether a set is succ-exchangeable (pred-exchangeable) in polynomial time.*

The idea of the proof is to construct a function $f$ that encodes each non-exchangeable set by a subset of $K$ no larger than $W_1$. To show this encoding is injective, we provide a decoding function $g$ and show that $g \circ f$ is an identity on non-exchangeable sets.

*Proof.* As in Observation 2.12, the proofs for *succ*- and *pred*-exchangeable sets are analogous, but for the sake or clarity we include both proofs in full detail.

Non-*succ*-exchangeable sets. For any set $Y \subseteq K$ we define the function $f_Y : W_1 \to K \cup \{\text{nil}\}$ as follows: for any element $w \in W_1$ we define $f_Y(w)$ (the *least expensive predecessor of $w$ outside $Y$*) to be the element of $(K \setminus Y) \cap pred(w)$ which has the smallest processing time, or nil if $(K \setminus Y) \cap pred(w)$ is empty. We now take $f(Y)$ (the set of the *least expensive predecessors outside $Y$*) to be the set $\{f_Y(w) : w \in W_1\} \setminus \{\text{nil}\}$. We see that $f(Y)$ is indeed a set of cardinality at most $|W_1|$.

Now we aim to prove that $f$ is injective on the family of non-*succ*-exchangeable sets. To this end we define the reverse function $g$. For a set $Z \subseteq K$ (which we think of as the set of the least expensive predecessors outside some $Y$) let $g(Z)$ be the set of such elements $v$ of $K$ that there exists $w \in succ(v)$ such that for any $z_w \in Z \cap pred(w)$ we have $t(z_w) > t(v)$. Notice, in particular, that $g(Z) \cap Z = \emptyset$, as for $v \in Z$ and $w \in succ(v)$ we have $v \in Z \cap pred(w)$.

First we prove $g(f(Y)) \subseteq Y$ for any $Y \subseteq K$. Take any $v \in K \setminus Y$ and consider any $w \in succ(v)$. Then $f_Y(w) \neq \text{nil}$ and $t(f_Y(w)) \leq t(v)$, as $v \in (K \setminus Y) \cap pred(w)$. Thus $v \notin g(f(Y))$, as for any $w \in succ(v)$ we can take a witness $z_w = f_Y(w)$ in the definition of $g(f(Y))$.

In the other direction, let us assume that $Y$ does not satisfy $Y \subseteq g(f(Y))$. This means we have $u \in Y \setminus g(f(Y))$. Then we show that $Y$ is *succ*-

exchangeable. Consider any $w \in succ(u)$. As $u \notin g(f(Y))$, there exists $z_w \in f(Y) \cap pred(w)$ with $t(z_w) \leq t(u)$. But $f(Y) \cap Y = \emptyset$, while $u \in Y$; and as all the values of $t$ are distinct, $t(z_w) < t(u)$ and $z_w$ satisfies the condition for $v_w$ in the definition of *succ*-exchangeability.

Non-*pred*-exchangeable sets. For any set $Y \subseteq K$ we define the function $f_Y : W_1 \to K \cup \{\text{nil}\}$ as follows: for any element $w \in W_1$ we define $f_Y(w)$ (the *most expensive successor of $w$ in $Y$*) to be the element of $Y \cap succ(w)$ which has the largest processing time, or nil if $Y \cap succ(w)$ is empty. We now take $f(Y)$ (the set of the *most expensive successors in $Y$*) to be the set $\{f_Y(w) : w \in W_1\} \setminus \{\text{nil}\}$. We see that $f(Y)$ is indeed a set of cardinality at most $|W_1|$.

Now we aim to prove that $f$ is injective on the family of non-*pred*-exchangeable sets. To this end we define the reverse function $g$. For a set $Z \subseteq K$ (which we think of as the set of most expensive successors in some $Y$) let $g(Z)$ be the set of such elements $v$ of $K$ that for any $w \in pred(v)$ there exists a $z_w \in Z \cap succ(w)$ with $t(z_w) \geq t(v)$. Notice, in particular, that $g(Z) \subseteq Z$, as for $v \in Z$ the job $z_w = v$ is a good witness for any $w \in pred(v)$.

First we prove $Y \subseteq g(f(Y))$ for any $Y \subseteq K$. Take any $v \in Y$ and consider any $w \in pred(v)$. Then $f_Y(w) \neq \text{nil}$ and $t(f_Y(w)) \geq t(v)$, as $v \in Y \cap succ(w)$. Thus $v \in g(f(Y))$, as for any $w \in pred(v)$ we can take $z_w = f_Y(w)$ in the definition of $g(f(Y))$.

In the other direction, let us assume that $Y$ does not satisfy $g(f(Y)) \subseteq Y$. This means we have $v \in g(f(Y)) \setminus Y$. Then we show that $Y$ is *pred*-exchangeable. Consider any $w \in pred(v)$. As $v \in g(f(Y))$ there exists $z_w \in f(Y) \cap succ(w)$ with $t(z_w) \geq t(v)$. But $f(Y) \subseteq Y$, while $v \notin Y$; and as all the values of $t$ are distinct, $t(z_w) > t(v)$ and $z_w$ satisfies the condition for $u_w$ in the definition of *pred*-exchangeability.

Thus, in both cases, if $Y$ is non-exchangeable then $g(f(Y)) = Y$ (in fact it is possible to prove in both cases that $Y$ is non-exchangeable iff $g(f(Y)) = Y$). As there are $\sum_{l=0}^{|W_1|} \binom{|K|}{l}$ possible values of $f(Y)$, the first part of the lemma is proven. For the second, it suffices to notice that *succ*- and *pred*-exchangeability can be checked in time $O(|K|^2|W_1|)$ directly from the definition. □

**Example 2.14.** To illustrate the applicability of Lemma 2.13, we analyze the following very simple case: assume the whole set $W_1 \setminus \{v_{begin}\}$ succeeds $I_1$, i.e., for every $w \in W_1 \setminus \{v_{begin}\}$ and $v \in I_1$ we have $w \not\prec v$. If $\varepsilon_1$ is small, then we can use the first case of Observation 2.12 for the whole set $K = I_1$:

we have $pred(K) = \{v_{begin}\}$ and we only look for orderings that put $v_{begin}$ as the first processed job. Thus, we can apply Proposition 2.9 with algorithm $\mathcal{R}$ that rejects sets $X \subseteq V$ where $X \cap I_1$ is *succ*-exchangeable with respect to $I_1$. By Lemma 2.13, the number of sets accepted by $\mathcal{R}$ is bounded by $2^{|W_1|} \sum_{l \leq |W_1|} \binom{|I_1|}{l}$, which is small if $|W_1| \leq \varepsilon_1 n$.

### 2.4.6 Important jobs at $n/2$

As was already mentioned in the overview, the assumptions of Observation 2.12 are quite strict; therefore, we need to learn a bit more on how $\sigma_{opt}$ behaves on $W_1$ in order to distinguish a suitable place for an application. As $|W_1| \leq 2\varepsilon_1 n$, we can afford branching into few subcases for every job in $W_1$.

Let $A = \{1, 2, \ldots, n/4\}$, $B = \{n/4{+}1, \ldots, n/2\}$, $C = \{n/2{+}1, \ldots, 3n/4\}$, $D = \{3n/4 + 1, \ldots, n\}$, i.e., we split $\{1, 2, \ldots, n\}$ into quarters. For each $w \in W_1 \setminus \{v_{begin}, v_{end}\}$ we branch into four cases: whether $\sigma_{opt}(w)$ belongs to $A$, $B$, $C$ or $D$. This branching leads to $4^{|W_1|-2} \leq 2^{4\varepsilon_1 n}$ subcases, and thus the same overhead in the time complexity. Of course, we already know that $\sigma_{opt}(v_{begin}) \in A$ and $\sigma_{opt}(v_{end}) \in D$. We terminate all the branches such that the guesses about alignment of jobs from $W_1$ contradict precedence constraints inside $W_1$.

Now consider a fixed branch. For any $\Gamma \in \{A, B, C, D\}$ let $W_1^\Gamma$ be the set of elements of $W_1$ to be placed in $\Gamma$. Moreover let $W_1^{AB} = W_1^A \cup W_1^B$ and $W_1^{CD} = W_1^C \cup W_1^D$.

Let us now see what we can learn in a fixed branch about the behaviour of $\sigma_{opt}$ on $I_1$. Let

$$W_2^{AB} = \left\{v \in I_1 : \exists_w \ \left(w \in W_1^{AB} \wedge v < w\right)\right\}$$
$$W_2^{CD} = \left\{v \in I_1 : \exists_w \ \left(w \in W_1^{CD} \wedge w < v\right)\right\},$$

that is $W_2^{AB}$ (resp. $W_2^{CD}$) are those elements of $I_1$ which are forced into the first (resp. second) half of $\sigma_{opt}$ by the choices we made about $W_1$. If one of the $W_2$ sets is significantly larger than $W_1$, we have obtained a gain — by branching into $2^{4\varepsilon_1 n}$ branches we gained additional information about a significant number of other elements (and so we will be able to avoid considering a significant number of sets in the DP algorithm). This is formalized in the following lemma:

**Lemma 2.15.** *Consider a fixed branch and let $0 < \alpha < 1/2$ be a fixed constant. If $W_2^{AB}$ or $W_2^{CD}$ has at least $\varepsilon_2 n$ elements, then the DP algorithm*

*can be augmented to solve the instance in the considered branch in time*

$$T_2(n) = \left( \binom{n}{(1/2 - \alpha\varepsilon_2)n} + 2^{(1-\varepsilon_2)n} \binom{\varepsilon_2 n}{\alpha\varepsilon_2 \cdot n} \right) n^{O(1)}.$$

*Proof.* We describe here only the case $|W_2^{AB}| \geq \varepsilon_2 n$. The second case is symmetrical.

Recall that the set $W_2^{AB}$ needs to be placed in $A \cup B$ by the optimal ordering $\sigma_{opt}$. We use Proposition 2.9 with an algorithm $\mathcal{R}$ that accepts the following sets $X \subseteq V$:

1. all sets $X$ of size at most $n/2 - \alpha|W_2^{AB}|$, there are at most $\binom{n}{(1/2-\alpha\varepsilon_2)n} n$ such sets;

2. among sets $X$ of size $n/2 - \alpha|W_2^{AB}| \leq |X| \leq n/2$, only those sets for which $|W_2^{AB} \setminus X| \leq \alpha|W_2^{AB}|$, there are at most $2^{(1-\varepsilon_2)n}\binom{\varepsilon_2 n}{\alpha\varepsilon_2 \cdot n} n$ such sets;

3. among sets $X$ of size at least $n/2$, only the sets containing $W_2^{AB}$, there are at most $2^{(1-\varepsilon_2)n}$ such sets.

Moreover, the algorithm $\mathcal{R}$ tests if the set $X$ conforms with the guessed sets $W_1^\Gamma$ for $\Gamma \in \{A, B, C, D\}$, i.e.:

$$\begin{aligned}
|X| \leq n/4 &\Rightarrow (W_1^B \cup W_1^C \cup W_1^D) \cap X = \emptyset \\
n/4 \leq |X| \leq n/2 &\Rightarrow (W_1^A \subseteq X \wedge (W_1^C \cup W_1^D) \cap X = \emptyset) \\
n/2 \leq |X| \leq 3n/4 &\Rightarrow ((W_1^A \cup W_1^B) \subseteq X \wedge W_1^D \cap X = \emptyset) \\
3n/4 \leq |X| &\Rightarrow (W_1^A \cup W_1^B \cup W_1^C) \subseteq X.
\end{aligned}$$

We now verify that $\sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$ for any $1 \leq i \leq n$. For $i \leq n/2 - \alpha|W_2^{AB}|$ it is obvious, and for $i \geq n/2$ it follows from the fact that $W_2^{AB}$ is placed in $A \cup B$ by $\sigma_{opt}$. Finally, for $n/2 - \alpha|W_2^{AB}| \leq i \leq n/2$ we have

$$|W_2^{AB} \setminus \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})| \leq n/2 - i \leq \alpha|W_2^{AB}|.$$

The bound $T_2(n)$ is immediate from the discussion above. □

Note that we have $2^{4\varepsilon_1 n}$ overhead so far, due to guessing placement of the jobs from $W_1$. By Lemma 2.1, $\binom{\varepsilon_2 n}{\alpha\varepsilon_2 n} = O((2 - c(\alpha))^{\varepsilon_2 n})$ and $\binom{n}{(1/2-\alpha\varepsilon_2)n} = O((2 - c(\alpha, \varepsilon_2))^n)$, for some positive constants $c(\alpha)$ and $c(\alpha, \varepsilon_2)$ that depend

only on $\alpha$ and only on $\alpha$ and $\varepsilon_2$, respectively. Thus, for any small fixed $\varepsilon_2$ and any fixed $0 < \alpha < 1/2$ we can choose $\varepsilon_1$ sufficiently small so that $2^{4\varepsilon_1 n} T_2(n) = O(c^n)$ for some $c < 2$. Note that $2^{4\varepsilon_1 n} T_2(n)$ is an upper bound on the total time spent on processing all the considered subcases.

Let $W_2 = W_2^{AB} \cup W_2^{CD}$ and $I_2 = I_1 \setminus W_2$. From this point we assume that $|W_2^{AB}|, |W_2^{CD}| \leq \varepsilon_2 n$, hence $|W_2| \leq 2\varepsilon_2 n$ and $|I_2| \geq (1 - 2\varepsilon_1 - 2\varepsilon_2)n$. For each $v \in W_2^{AB}$ we branch into two subcases, whether $\sigma_{opt}(v)$ belongs to $A$ or $B$. Similarly, for each $v \in W_2^{CD}$ we guess whether $\sigma_{opt}(v)$ belongs to $C$ or $D$. Again, we terminate branches which are trivially contradicting the constraints. This step gives us an additional $2^{|W_2|} \leq 2^{2\varepsilon_2 n}$ overhead in the time complexity. We denote the set of elements of $W_2$ assigned to quarter $\Gamma \in \{A, B, C, D\}$ by $W_2^\Gamma$.

### 2.4.7 Quarters and applications of the core lemma

In this section we try to apply Lemma 2.13 as follows: We look which elements of $I_2$ can be placed in $A$ (the set $P^A$) and which cannot (the set $P^{\neg A}$). Similarly we define the set $P^D$ (can be placed in $D$) and $P^{\neg D}$ (cannot be placed in $D$). For each of these sets, we try to apply Lemma 2.13 to some subset of it. If we fail, then in the next subsection we infer that the solutions in the quarters are partially independent of each other, and we can solve the problem in time roughly $O(2^{3n/4})$. Let us now proceed with a more detailed argumentation.

We define the following two partitions of $I_2$:

$$\begin{aligned}
P^{\neg A} &= \left\{ v \in I_2 : \exists_w \left( w \in W_1^B \wedge w < v \right) \right\}, \\
P^A &= I_2 \setminus P^{\neg A} = \left\{ v \in I_2 : \forall_w \left( w < v \Rightarrow w \in W_1^A \right) \right\}, \\
P^{\neg D} &= \left\{ v \in I_2 : \exists_w \left( w \in W_1^C \wedge w > v \right) \right\}, \\
P^D &= I_2 \setminus P^{\neg D} = \left\{ v \in I_2 : \forall_w \left( w > v \Rightarrow w \in W_1^D \right) \right\}.
\end{aligned}$$

In other words, the elements of $P^{\neg A}$ cannot be placed in $A$ because some of their requirements are in $W_1^B$, and the elements of $P^{\neg D}$ cannot be placed in $D$ because they are required by some elements of $W_1^C$. Note that these definitions are independent of $\sigma_{opt}$, so sets $P^\Delta$ for $\Delta \in \{A, \neg A, \neg D, D\}$ can

be computed in polynomial time. Let

$$\begin{aligned} p^A &= |\sigma_{opt}(P^A) \cap A|, \\ p^B &= |\sigma_{opt}(P^{\neg A}) \cap B|, \\ p^C &= |\sigma_{opt}(P^{\neg D}) \cap C|, \\ p^D &= |\sigma_{opt}(P^D) \cap D|. \end{aligned}$$

Note that $p^\Gamma \leq n/4$ for every $\Gamma \in \{A, B, C, D\}$. As $p^A = n/4 - |W_1^A \cup W_2^A|$, $p^D = n/4 - |W_1^D \cup W_2^D|$, these values can be computed by the algorithm. We branch into $(1 + n/4)^2$ further subcases, guessing the (still unknown) values $p^B$ and $p^C$.

Let us focus on the quarter $A$ and assume that $p^A$ is significantly smaller than $|P^A|/2$. We claim that we can apply Lemma 2.13 as follows. While computing $\sigma[X]$, if $|X| \geq n/4$, we can represent $X \cap P^A$ as a disjoint sum of two subsets $X_A^A, X_{BCD}^A \subseteq P^A$. The first one is of size $p^A$, and represents the elements of $X \cap P^A$ placed in quarter $A$, and the second represents the elements of $X \cap P^A$ placed in quarters $B \cup C \cup D$. Note that the elements of $X_{BCD}^A$ have all predecessors in the quarter $A$, so by Observation 2.12 the set $X_{BCD}^A$ has to be non-*succ*-exchangeable with respect to $P^A \setminus X_A^A$; therefore, by Lemma 2.13, we can consider only a very narrow choice of $X_{BCD}^A$. Thus, the whole part $X \cap P^A$ can be represented by its subset of cardinality at most $p^A$ plus some small information about the rest. If $p^A$ is significantly smaller than $|P^A|/2$, this representation is more concise than simply remembering a subset of $P^A$. Thus we obtain a better bound on the number of feasible sets.

A symmetric situation arises when $p^D$ is significantly smaller than $|P^D|/2$; moreover, we can similarly use Lemma 2.13 if $p^B$ is significantly smaller than $|P^{\neg A}|/2$ or $p^C$ than $|P^{\neg D}|/2$. This is formalized by the following lemma.

**Lemma 2.16.** *If $p^\Gamma < |P^\Delta|/2$ for some $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$ and $\varepsilon_1 \leq 1/4$, then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_p(n) = 2^{n - |P^\Delta|} \binom{|P^\Delta|}{p^\Gamma} \binom{n}{|W_1|} n^{O(1)}.$$

*Proof.* We describe here only the case $\Delta = \Gamma = A$, the other cases are analogous.

On a high-level, we want to proceed as in Proposition 2.9, i.e., use the standard DP algorithm described in Section 2.4.1, while terminating the

computation for some unfeasible subsets of $V$. However, in this case we need to slightly modify the recursive formula used in the computations, and we compute $\sigma[X, L]$ for $X \subseteq V$, $L \subseteq X \cap P^A$. Intuitively, the set $X$ plays the same role as before, whereas $L$ is the subset of $X \cap P^A$ that was placed in the quarter $A$. Formally, $\sigma[X, L]$ is the ordering of $X$ that attains the minimum total cost among those orderings $\sigma$ for which $L = P^A \cap \sigma^{-1}(A)$. Thus, in the DP algorithm we use the following recursive formula:

$$T(\sigma[X,L]) = \begin{cases} \min_{v \in \max(X)} [T(\sigma[X \setminus \{v\}, L \setminus \{v\}]) + T(v, |X|)] & \\ \qquad\qquad \text{if } |X| \leq n/4 \text{ and } L = X \cap P^A, & \\ +\infty \qquad\qquad \text{if } |X| \leq n/4 \text{ and } L \neq X \cap P^A, & \\ \min_{v \in \max(X) \setminus L} [T(\sigma[X \setminus \{v\}, L]) + T(v, |X|)] & \\ \qquad\qquad \text{otherwise.} & \end{cases}$$

In the next paragraphs we describe a polynomial-time algorithm $\mathcal{R}$ that accepts or rejects pairs of subsets $(X, L)$, $X \subseteq V$, $L \subseteq X \cap P^A$; we terminate the computation on rejected pairs $(X, L)$. As each single calculation of $\sigma[X, L]$ uses at most $|X|$ recursive calls, the time complexity of the algorithm is bounded by the number of accepted pairs, up to a polynomial multiplicative factor. We now describe the algorithm $\mathcal{R}$.

First, given a pair $(X, L)$, we ensure that we fulfill the guessed sets $W_k^\Gamma$, $\Gamma \in \{A, B, C, D\}$, $k = 1, 2$. E.g., we require $W_k^B \subseteq X$ if $|X| \geq n/2$ and $W_k^B \cap X = \emptyset$ if $|X| \leq n/4$. We require similar conditions for other quarters $A$, $C$ and $D$ (cf. proof of Lemma 2.15). Moreover, we require that $X$ is downward closed. Note that this implies $X \cap P^{\neg A} = \emptyset$ if $|X| \leq n/4$ and $P^{\neg D} \subseteq X$ if $|X| \geq 3n/4$.

Second, we require the following:

1. If $|X| \leq n/4$, we require that $L = X \cap P^A$ and $|L| \leq p^A$; as $p^A \leq |P^A|/2$, there are at most $2^{n-|P^A|}\binom{|P^A|}{p^A} n$ such pairs $(X, L)$;

2. Otherwise, we require that $|L| = p^A$ and that the set $X \cap (P^A \setminus L)$ is non-*succ*-exchangeable with respect to $P^A \setminus L$; by Lemma 2.13 there are at most $\sum_{l \leq |W_1|} \binom{|P^A \setminus L|}{l} \leq n\binom{n}{|W_1|}$ (since $|W_1| \leq 2\varepsilon_1 n \leq n/2$) non-*succ*-exchangeable sets with respect to $P^A \setminus L$, thus there are at most $2^{n-|P^A|}\binom{|P^A|}{p^A}\binom{n}{|W_1|} n$ such pairs $(X, L)$.

Let us now check the correctness of the above pruning. Let $0 \leq i \leq n$ and let $X = \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ and $L = \sigma_{opt}^{-1}(A) \cap X \cap P^A$. It is easy to see

that Observation 2.12 implies that in case $i \geq n/4$ the set $X \cap (P^A \setminus L)$ is non-*succ*-exchangeable and the pair $(X, L)$ is accepted.

The cases $(\Gamma, \Delta) \in \{(B, \neg A), (C, \neg D), (D, D)\}$ are analogous: $L$ corresponds to jobs from $P^\Delta$ scheduled to be done in segment $\Gamma$ and we require that $X \cap (P^\Delta \setminus L)$ is non-*pred*-exchangeable (instead of non-*succ*-exchangeable) in case $\Delta = \neg D, D$. The recursive definition of $T(\sigma[X, L])$ should be also adjusted. □

Observe that if any of the sets $P^\Delta$ for $\Delta \in \{A, \neg A, \neg D, D\}$ is significantly larger than $n/2$, one of the situations in Lemma 2.16 indeed occurs, since $p^\Gamma \leq n/4$ for $\Gamma \in \{A, B, C, D\}$ and $|W_1|$ is small.

**Lemma 2.17.** *If $2\varepsilon_1 < 1/4 + \varepsilon_3/2$ and at least one of the sets $P^A$, $P^{\neg A}$, $P^{\neg D}$ and $P^D$ is of size at least $(1/2 + \varepsilon_3)n$, then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_3(n) = 2^{(1/2-\varepsilon_3)n} \binom{(1/2+\varepsilon_3)n}{n/4} \binom{n}{2\varepsilon_1 n} n^{O(1)}.$$

*Proof.* The claim is straightforward; note only that the term $2^{n-|P^\Delta|}\binom{|P^\Delta|}{p^\Gamma}$ for $p^\Gamma < |P^\Delta|/2$ is a decreasing function of $|P^\Delta|$. □

Note that we have $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)}$ overhead so far. As $\binom{(1/2+\varepsilon_3)n}{n/4} = O((2 - c(\varepsilon_3))^{(1/2+\varepsilon_3)n})$ for some constant $c(\varepsilon_3) > 0$, for any small fixed $\varepsilon_3$ we can choose sufficiently small $\varepsilon_2$ and $\varepsilon_1$ to have $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)} T_3(n) = O(c^n)$ for some $c < 2$.

From this point we assume that $|P^A|, |P^{\neg A}|, |P^{\neg D}|, |P^D| \leq (1/2 + \varepsilon_3)n$. As $P^A \cup P^{\neg A} = I_2 = P^{\neg D} \cup P^D$ and $|I_2| \geq (1 - 2\varepsilon_1 - 2\varepsilon_2)n$, this implies that these four sets are of size at least $(1/2 - 2\varepsilon_1 - 2\varepsilon_2 - \varepsilon_3)n$, i.e., they are of size roughly $n/2$. Having bounded the sizes of the sets $P^\Delta$ from below, we are able to use Lemma 2.16 again: if any of the numbers $p^A$, $p^B$, $p^C$, $p^D$ is significantly smaller than $n/4$, then it is also significantly smaller than half of the cardinality of the corresponding set $P^\Delta$.

**Lemma 2.18.** *Let $\varepsilon_{123} = 2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_3$. If at least one of the numbers $p^A$, $p^B$, $p^C$ and $p^D$ is smaller than $(1/4 - \varepsilon_4)n$ and $\varepsilon_4 > \varepsilon_{123}/2$, then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_4(n) = 2^{(1/2+\varepsilon_{123})n} \binom{(1/2-\varepsilon_{123})n}{(1/4-\varepsilon_4)n} \binom{n}{2\varepsilon_1 n} n^{O(1)}.$$

*Proof.* As, before, the claim is a straightforward application of Lemma 2.16, and the fact that the term $2^{n-|P^\Delta|}\binom{|P^\Delta|}{p^\Gamma}$ for $p^\Gamma < |P^\Delta|/2$ is a decreasing function of $|P^\Delta|$. □

So far we have $2^{(4\varepsilon_1+2\varepsilon_2)n}n^{O(1)}$ overhead. Similarly as before, for any small fixed $\varepsilon_4$ if we choose $\varepsilon_1, \varepsilon_2, \varepsilon_3$ sufficiently small, we have $\binom{(1/2-\varepsilon_{123})n}{(1/4-\varepsilon_4)n} = O((2-c(\varepsilon_4))^{(1/2-\varepsilon_{123})n})$ and $2^{(4\varepsilon_1+2\varepsilon_2)n}n^{O(1)}T_4(n) = O(c^n)$ for some $c < 2$.

Thus we are left with the case when $p^A, p^B, p^C, p^D \geq (1/4-\varepsilon_4)n$.

### 2.4.8 The remaining case

In this subsection we infer that in the remaining case the quarters $A$, $B$, $C$ and $D$ are somewhat independent, which allows us to develop a faster algorithm. More precisely, note that $p^\Gamma \geq (1/4-\varepsilon_4)n$, $\Gamma \in \{A, B, C, D\}$, means that almost all elements that are placed in $A$ by $\sigma_{opt}$ belong to $P^A$, while almost all elements placed in $B$ belong to $P^{\neg A}$. Similarly, almost all elements placed in $D$ belong to $P^D$ and almost all elements placed in $C$ belong to $P^{\neg D}$. As $P^A \cap P^{\neg A} = \emptyset$ and $P^{\neg D} \cap P^D = \emptyset$, this implies that what happens in the quarters $A$ and $B$, as well as $C$ and $D$, is (almost) independent. This key observation can be used to develop an algorithm that solves this special case in time roughly $O(2^{3n/4})$.

Let $W_3^B = I_2 \cap (\sigma_{opt}^{-1}(B) \setminus P^{\neg A})$ and $W_3^C = I_2 \cap (\sigma_{opt}^{-1}(C) \setminus P^{\neg D})$. As $p^B, p^C \geq (1/4-\varepsilon_4)n$ we have that $|W_3^B|, |W_3^C| \leq \varepsilon_4 n$. We branch into at most $n^2\binom{n}{\varepsilon_4 n}^2$ subcases, guessing the sets $W_3^B$ and $W_3^C$. Let $W_3 = W_3^B \cup W_3^C$, $I_3 = I_2 \setminus W_3$, $Q^\Delta = P^\Delta \setminus W_3$ for $\Delta \in \{A, \neg A, \neg D, D\}$. Moreover, let $W^\Gamma = W_1^\Gamma \cup W_2^\Gamma \cup W_3^\Gamma$ for $\Gamma \in \{A, B, C, D\}$, using the convention $W_3^A = W_3^D = \emptyset$.

Note that in the current branch for any ordering and any $\Gamma \in \{A, B, C, D\}$, the segment $\Gamma$ gets all the jobs from $W^\Gamma$ and $q^\Gamma = n/4 - |W^\Gamma|$ jobs from appropriate $Q^\Delta$ ($\Delta = A, \neg A, \neg D, D$ for $\Gamma = A, B, C, D$, respectively). Thus, the behaviour of an ordering $\sigma$ in $A$ influences the behaviour of $\sigma$ in $C$ by the choice of which elements of $Q^A \cap Q^{\neg D}$ are placed in $A$, and which in $C$. Similar dependencies are between $A$ and $D$, $B$ and $C$, as well as $B$ and $D$ (see Figure 2.5). In particular, there are no dependencies between $A$ and $B$, as well as $C$ and $D$, and we can compute the optimal arrangement by keeping track of only three out of four dependencies at once, leading us to an algorithm running in time roughly $O(2^{3n/4})$. This is formalized in the following lemma:

**Lemma 2.19.** *If $2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_4 < 1/4$, the remaining case can be solved by an algorithm running in time bounded by*

$$T_5(n) = \binom{n}{\varepsilon_4 n}^2 2^{(3/4+\varepsilon_3)n} n^{O(1)}.$$

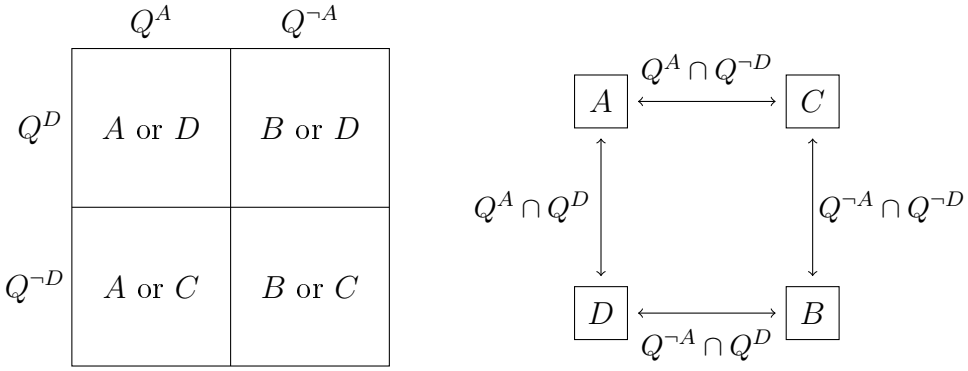


Figure 2.5: Dependencies between quarters and sets $Q^\Delta$. The left part of the figure illustrates where the jobs from $Q^{\Delta_1} \cap Q^{\Delta_2}$ may be placed. The right part of the figure illustrates the dependencies between the quarters.

*Proof.* Let $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$. For each set $Y \subseteq Q^\Delta$ of size $q^\Gamma$, for each bijection (partial ordering) $\sigma^\Gamma(Y) : Y \cup W^\Gamma \to \Gamma$ let us define its cost as

$$T(\sigma^\Gamma(Y)) = \sum_{v \in Y \cup W^\Gamma} T(v, \sigma^\Gamma(Y)(v)).$$

Let $\sigma^\Gamma_{opt}(Y)$ be the partial ordering that minimizes the cost (recall that it is unique due to the initial steps in Section 2.4.4). Note that if we define $Y^\Gamma_{opt} = \sigma^{-1}_{opt}(\Gamma) \cap Q^\Delta$ for $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$, then the ordering $\sigma_{opt}$ consists of the partial orderings $\sigma^\Gamma_{opt}(Y^\Gamma_{opt})$.

We first compute the values $\sigma^\Gamma_{opt}(Y)$ for all $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$ and $Y \subseteq Q^\Delta$, $|Y| = q^\Gamma$, by a straightforward modification of the DP algorithm. For fixed pair $(\Gamma, \Delta)$, the DP algorithm computes $\sigma^\Gamma_{opt}(Y)$ for all $Y$ in time

$$2^{|W^\Gamma|+|Q^\Delta|} n^{O(1)} \leq 2^{(2\varepsilon_1+2\varepsilon_2+\varepsilon_4)n+(1/2+\varepsilon_3)n} n^{O(1)} = O(2^{(3/4+\varepsilon_3)n}).$$

The last inequality follows from the assumption $2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_4 < 1/4$.

Let us focus on the sets $Q^A \cap Q^{\neg D}$, $Q^A \cap Q^D$, $Q^{\neg A} \cap Q^{\neg D}$ and $Q^{\neg A} \cap Q^D$. Without loss of generality we assume that $Q^A \cap Q^{\neg D}$ is the smallest among those. As they all are pairwise disjoint and sum up to $I_2$, we have $|Q^A \cap Q^{\neg D}| \leq n/4$. We branch into at most $2^{|Q^A \cap Q^{\neg D}| + |Q^{\neg A} \cap Q^D|}$ subcases, guessing the sets

$$Y^{AC}_{opt} = Y^A_{opt} \cap (Q^A \cap Q^{\neg D}) = (Q^A \cap Q^{\neg D}) \setminus Y^C_{opt} \quad \text{and}$$
$$Y^{BD}_{opt} = Y^B_{opt} \cap (Q^{\neg A} \cap Q^D) = (Q^{\neg A} \cap Q^D) \setminus Y^D_{opt}.$$

Then, we choose the set

$$Y^{AD}_{opt} = Y^A_{opt} \cap (Q^A \cap Q^D) = (Q^A \cap Q^D) \setminus Y^D_{opt}$$

that optimizes

$$T(\sigma^A_{opt}(Y^{AC}_{opt} \cup Y^{AD}_{opt})) + T(\sigma^D_{opt}(Q^D \setminus (Y^{AD}_{opt} \cup Y^{BD}_{opt})).$$

Independently, we choose the set

$$Y^{BC}_{opt} = Y^B_{opt} \cap (Q^{\neg A} \cap Q^{\neg D}) = (Q^{\neg A} \cap Q^{\neg D}) \setminus Y^C_{opt}$$

that optimizes

$$T(\sigma^B_{opt}(Y^{BC}_{opt} \cup Y^{BD}_{opt})) + T(\sigma^C_{opt}(Q^{\neg D} \setminus (Y^{BC}_{opt} \cup Y^{AC}_{opt})).$$

To see the correctness of the above step, note that $Y^A_{opt} = Y^{AC}_{opt} \cup Y^{AD}_{opt}$, and similarly for other quarters.

The time complexity of the above step is bounded by

$$\begin{aligned} &2^{|Q^A \cap Q^{\neg D}| + |Q^{\neg A} \cap Q^D|} \left(2^{|Q^A \cap Q^D|} + 2^{|Q^{\neg A} \cap Q^{\neg D}|}\right) n^{O(1)} \\ &\quad = 2^{|Q^A \cap Q^{\neg D}|} \left(2^{|Q^D|} + 2^{|Q^{\neg A}|}\right) n^{O(1)} \\ &\quad \leq 2^{(3/4+\varepsilon_3)n} n^{O(1)} \end{aligned}$$

and the bound $T_5(n)$ follows. □

So far we have $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)}$ overhead. For sufficiently small $\varepsilon_4$ we have $\binom{n}{\varepsilon_4 n} = O(2^{n/16})$ and then for sufficiently small constants $\varepsilon_k$, $k = 1, 2, 3$ we have $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)} T_5(n) = O(c^n)$ for some $c < 2$.

### 2.4.9 Numerical values of the constants

Before we give the values of the constants $\varepsilon_k$ and $\alpha$ (used in Lemma 2.15), let us make a small optimization. Note that when invoking Lemma 2.15, we only need to know how $\sigma_{opt}$ splits the set $W_1$ between halves $A \cup B$ and $C \cup D$, i.e., we need to know the sets $W_1^{AB}$ and $W_1^{CD}$ instead of the whole quadruple $W_1^{\Gamma}$ for $\Gamma \in \{A, B, C, D\}$. This leads to $2^{2\varepsilon_1 n}$ overhead (instead of $2^{4\varepsilon_1 n}$) in front of the bound $T_2(n)$. Using this observation and the following values of the constants:

$$\begin{aligned}
\varepsilon_1 &= 9.98046875 \cdot 10^{-16} \\
\varepsilon_2 &= 0.00002246093750077387619018554687 5 \\
\varepsilon_3 &= 0.00701843070983939668794721364974 9755859375 \\
\varepsilon_4 &= 0.01652666037343616678841463726712390780448 \\
\alpha &= 0.4976413249969482421875
\end{aligned}$$

we get that the running time of our algorithm is bounded by:

$$O\left(\left(2 - 5 \cdot 10^{-16}\right)^n\right).$$

# Chapter 3

# Fixed-parameter algorithms and kernelization

## 3.1 Introduction

Let us start with the following motivating example, described in the textbook of Downey and Fellows [56]. Assume we have collected many data points from an experiment. However, it appears that some data points are contradicting another. We want to select as small as possible number of data points such that after discarding this set no contradictions remain. By replacing data points with vertices, and pairs of contradicting data points by edges, we arrive at the well-known Vertex Cover problem:

| Vertex Cover |
| --- |
| **Input:** An undirected graph $G = (V, E)$ and an integer $k$. |
| **Parameter:** $k$. |
| **Question:** Does there exist a set $X \subseteq V$ of at most $k$ vertices, such that each edge in $G$ has an endpoint in $X$? |

Although Vertex Cover is NP-complete, the fact that the number of data points to be discarded — the parameter $k$ — is most probably very small, makes our situation not totally hopeless. As we will see later in this section, the Vertex Cover problem can be solved in $O(n2^k)$ time, feasible for $k$ around $20 - 30$ and even *very large* $n$ (the number of data points). Note that this would not be the case with the naive brute-force algorithm for Vertex Cover that runs in $O(n^k)$ time: even for $k = 20$, $O(n^{20})$ is not feasible for $n > 10$.

The aforementioned example is not a standalone one. If we are to solve an NP-hard problem with parameter $k$ that is assumed to be small (say, around 20), there is a huge difference between an algorithm running in time $O(nc^k)$ and $O(n^k)$: the first one is feasible for *almost arbitrarily large* values of $n$, when the second one is useless for $n \sim k$. Following Downey [57], let us illustrate this phenomenon by the example of the polynomial-time approximation schemes (PTAS). A typical PTAS finds an $(1+\varepsilon)$-approximate solution in time $O(n^{f(\varepsilon)})$ for some decreasing function $f$ (e.g., $f(\varepsilon) = 1/\varepsilon$), while some of them — called *efficient* PTASes (EPTAS) — run in time $O(f(\varepsilon)n^{O(1)})$. As pointed out by Downey [57], the PTASes of the first type are almost always not feasible for the allowed error of 20% (i.e., $\varepsilon = 0.2$), whereas the EPTASes may be useful for much smaller values of $\varepsilon$.

The difference between $O^\star(f(k))$ and $O(n^k)$ algorithms is the main motivation for the study of parameterized complexity. Let us recall that in the parameterized complexity setting, an instance comes with an integer parameter $k$ — formally, a parameterized problem $Q$ is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet $\Sigma$. We say that a problem is *fixed-parameter tractable* (*FPT*) if there exists an algorithm solving any instance $(x, k)$ in time $f(k)|x|^{O(1)}$ for some (usually exponential) computable function $f$. Intuitively, the parameter $k$ measures the hardness of the instance. Concluding the discussion on the examples at the beginning of this section, we consider fixed-parameter algorithms *efficient* for small values of $k$, as opposed to *inefficient* algorithms of time complexity $O(n^{f(k)})$.

Although over the last 20 years researchers developed a powerful toolbox for designing fixed-parameter algorithms, a few problems eluded their efforts. The most prominent two were Clique and Dominating Set, parameterized by the solution size.

Clique
**Input:** An undirected graph $G = (V, E)$ and an integer $k$.
**Parameter:** $k$.
**Question:** Does there a subgraph of $G$ that is isomorphic to a clique on $k$ vertices?

Dominating Set
**Input:** An undirected graph $G = (V, E)$ and an integer $k$.
**Parameter:** $k$.
**Question:** Does there exists a set of at most $k$ vertices that dominates all vertices of $G$?

The study of these two problems led to the development of the $W$-hierarchy of the parameterized problems:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P] \subseteq XP.$$

It turned out that CLIQUE is $W[1]$-complete, meaning that it is the hardest problem that can be expressed as verifying an existential formula of length bounded by a function of the parameter $k$. Similarly, DOMINATING SET is $W[2]$-complete, which contains all the problems expressible as a $\Sigma_2$ formula of length bounded in a function of the parameter. It is widely believed that $FPT \neq W[1]$ and the whole $W$-hierarchy is proper, giving the researchers tools to prove that some problems are not likely to admit a fixed-parameter algorithm.

Recall that a kernelization algorithm reduces in polynomial time the input instance $x$ with parameter $k$ to an equivalent one with the size bounded by $g(k)$ for some computable function $g$. We usually aim at polynomial kernels, that is, with function $g$ being a polynomial. Kernelization techniques can be viewed as polynomial time preprocessing routines for tackling NP-hard problems. In particular small (i.e. polynomial) kernels play an important role, and there are numerous positive results showing small kernels for various problems, including VERTEX COVER [32] and FEEDBACK VERTEX SET [126].

Note that if a problem $Q$ admits a kernelization algorithm and is decidable, it is also fixed-parameter tractable: we may simply reduce an input instance to a size bounded by a function of the parameter, and run a brute-force algorithm. Consider now a problem $Q$ that admits a fixed-parameter algorithm with running time $O(f(k)n^c)$ for a constant $c$ and a computable function $f$. Consider the following kernelization algorithm for $Q$: if $f(k) > n$, we do not alter the input instance; otherwise, the fixed-parameter algorithm solves the problem in polynomial, i.e., $O(n^{c+1})$, time. By this reasoning we obtained one of the core results of parameterized complexity: a decidable problem is FPT if and only if it is kernelizable.

It is sometimes convenient to consider problems with more than one parameter. In this case, we allow the function $f$ in the definition of a fixed-parameter algorithm, as well as the function $g$ that bounds the size of a kernel, to depend on all parameters. Note that, asymptotically, this is equivalent to consider a parameterized problem in the classical single-parameter setting where the single formal parameter is the sum of all parameters in the multi-parameter setting.

Before we dive into the depths of parameterized complexity, let us extend a bit the example of the Vertex Cover problem parameterized by the solution size. Consider the following branching algorithm for Vertex Cover (its idea can be traced back to Buss and Goldsmith [25]).

```
Function VertexCover(G, k)
1: if k < 0 then
2:     return NO
3: if E(G) = ∅ then
4:     return YES
5: let uv be an arbitrary edge of G
6: return VertexCover(G \ u, k − 1) ∨ VertexCover(G \ v, k − 1)
```

At each step, if $k \geq 0$ and $G$ still contains some edges, it chooses an arbitrary one and branches into two subcases, taking into the solution one of the endpoints of the chosen edge. As each recursive call of the VertexCover routine calls itself twice with the parameter decreased by one, the total running time of this algorithm is $O^\star(2^k)$ (and even $O(n2^k)$, if we take some care in the implementation). Thus the presented algorithm proves fixed-parameter tractability of Vertex Cover parameterized by the solution size. Currently the fastest parameterized algorithm for this problem, due to Chen et al. [33], works in $O^\star(1.2738^k)$ time.

Let us now focus on the kernelization of Vertex Cover. Consider the following kernelization algorithm due to Buss and Goldsmith [25]. The algorithm consists of three reduction rules. Each reduction rule, if applicable, takes an instance $(G, k)$ as an input and replaces it with an equivalent instance of smaller size. If no rule is applicable, we claim that the size of the resulting instance is bounded by $O(k^2)$.

1. **Isolated vertex rule.** If the input graph $G$ contains an isolated vertex, remove it.

2. **Large degree rule.** If the input graph $G$ contains a vertex of degree at least $k + 1$, remove it and decrease $k$ by one.

3. **Finishing rule.** If the large degree rule is not applicable and the graph contains more than $k^2$ edges, return a trivial NO-instance.

Let us first discuss the soundness of the above reduction rules. The first rule is obviously correct, as isolated vertices are not included in any inclusion-minimal vertex cover of $G$. If we have a vertex of degree at least $k + 1$, it

needs to be included in any vertex cover of size at most $k$, and the soundness of the second rule follows. Finally, if every vertex has degree bounded by $k$, a set of at most $k$ vertices can cover at most $k^2$ edges. This proves the correctness of the last rule.

Note that if no rule is applicable, we have at most $k^2$ edges in the graph $G$ and, since there are no isolated vertices, we have at most $2k^2$ vertices. We infer that Vertex Cover, parameterized by the solution size, admits a kernel of size $O(k^2)$. This bound was further improved and currently we know that any Vertex Cover instance can be reduced to an instance with at most $2k$ vertices [32].

After the above example, we are ready to present our results. In Section 3.2 we show a fixed-parameter algorithm for the Subset Feedback Vertex Set problem. Section 3.3 serves as a short introduction to the framework of kernelization lower bounds. Finally, in Section 3.4 we show our results on the hardness of polynomial kernelization of a few connectivity problems in graphs of bounded degeneracy.

## 3.2 Subset Feedback Vertex Set

Feedback Vertex Set (FVS) is one of the long–studied problems in the algorithms area. It can be stated as follows: given an undirected graph $G$ on $n$ vertices and a parameter $k$ decide if one can remove at most $k$ vertices from $G$ so that the remaining graph does not contain a cycle, i.e., is a forest. The problem of finding feedback sets in undirected graphs arises in a variety of applications in genetics, circuit testing, artificial intelligence, deadlock resolution, and analysis of manufacturing processes [62].

Because of its importance the feedback vertex set problem was studied from the approximation algorithms perspective in different variants and generalisations including Directed Feedback Vertex Set and Subset Feedback Vertex Set (see [61] and [63] for further references). In this section we study the Subset Feedback Vertex Set problem from the parametrized complexity perspective.

The long line of research concerning FVS in the parameterized complexity setting contains [8, 14, 27, 31, 50, 58, 87, 98, 117]. Currently the fastest known algorithm due to Cygan et al. works in $O^\star(3^k)$ time [42]. Thomassé [126] has shown a quadratic kernel for this problem improving previous results [18, 24]. The directed version has been proved to be FPT in 2008 by Chen et al. [35],

closing a long-standing open problem in the parameterized complexity community. The natural question concerning the parameterized complexity of the Subset Feedback Vertex Set problem was posed independently by Kawarabayashi at the 4th workshop on Graph Classes, Optimization, and Width Parameters (GROW 2009) and by Saurabh at the Dagstuhl seminar 'Parameterized complexity and approximation algorithms' (no. 09511, 2009) [52].

In the Subset Feedback Vertex Set problem (Subset-FVS) an instance comes with a subset of vertices $S$, and we ask for a set of at most $k$ vertices that hits all simple cycles passing through $S$. It is easy to see that Subset-FVS is a generalization of FVS by putting $S = V$. The weighted version of Subset-FVS was introduced by Even et al. [62] as a generalization of two problems: Feedback Vertex Set and Node Multiway Cut. Even et al. motivate Subset-FVS problem by explaining its applicability to genetic linkage.

Subset Feedback Vertex Set (Subset-FVS)
**Input:** An undirected graph $G = (V, E)$, a set $S \subseteq V$ and a positive integer $k$.
**Parameter:** $k$.
**Question:** Does there exist a set $T \subseteq V$ such that $|T| \leq k$ and no simple cycle in $G[V \setminus T]$ contains a vertex of $S$?

We also define a variant of Subset-FVS, where the set $S$ is a subset of edges of $G$.

Edge Subset Feedback Vertex Set (Edge-Subset-FVS)
**Input:** An undirected graph $G = (V, E)$, a set $S \subseteq E$ and a positive integer $k$.
**Parameter:** $k$.
**Question:** Does there exist a set $T \subseteq V$ with $|T| \leq k$, such that no simple cycle in $G[V \setminus T]$ contains an edge from $S$?

The two problems stated above are equivalent. To see this, note that if $(G, S, k)$ is an instance of Subset-FVS, we create an instance $(G, S', k)$ of Edge-Subset-FVS by selecting as $S'$ all the edges incident to any vertex of $S$. Then any simple cycle passing through a vertex of $S$ has to pass through an edge of $S'$, and conversely, any cycle passing through an edge of $S'$ contains a vertex from $S$. In the other direction, if $(G, S', k)$ is an instance of Edge-Subset-FVS, obtain $G'$ by replacing each edge $uv \in S'$ by a path

$u - x_{uv} - v$ of length 2, and solve the SUBSET-FVS instance $(G', S, k)$ where $S = \{x_e : e \in S'\}$. Clearly both reductions work in polynomial time and do not change the parameter. Thus, in the rest of this section we focus on solving EDGE SUBSET FEEDBACK VERTEX SET. A simple cycle containing an edge from $S$ is called an *$S$–cycle*.

It turns out that the SUBSET-FVS problem is very closely related to other graph separation problems, including NODE MULTIWAY CUT and NODE MULTICUT.

NODE MULTIWAY CUT
**Input:** An undirected graph $G = (V, E)$, a set of vertices $\mathcal{T} \subseteq V$, called *terminals*, and a positive integer $k$.
**Parameter:** $k$.
**Question:** Does there exist a set $T \subseteq V$ of at most $k$ non-terminals, such that no two terminals are in the same connected component of $G[V \setminus T]$?

NODE MULTICUT
**Input:** An undirected graph $G = (V, E)$, a set of pairs of vertices $\mathcal{T} \subseteq V \times V$, called *terminal pairs*, and a positive integer $k$.
**Parameter:** $k$.
**Question:** Does there exist a set $T \subseteq V$ of at most $k$ non-terminals, such that no terminal pair is contained in one connected component of $G[V \setminus T]$?

As observed by Even et al. [62] the weighted version of SUBSET-FVS is a generalization of NODE MULTIWAY CUT. It is straightforward to adjust their reduction to the unweighted parameterized case if we allow deletions of terminals; for sake of completeness, we include the reduction in Section 3.2.4.

The graph separation problems became an important part of the field of parameterized complexity after the discovery of the *important separators* technique by Marx [110]. This technique, after being improved by Chen et al. [34], is a core part of many fixed-parameter algorithms for various problems, including ALMOST-2-SAT [118] and (very recent) DIRECTED MULTIWAY CUT [36]. The fixed-parameter tractability for MULTICUT was independently obtained in 2010 by Bousquet et al. [21] and by Marx and Razgon [111].

The question of polynomial kernels for the graph separating problems is currently one of the most important open problems in the field of kernelization. Very recently Kratsch and Wahlström [103] applied matroid tools to

obtain a randomized polynomial kernel for the Odd Cycle Transversal problem. This result gives hope that the question of polynomial kernel of Node Multiway Cut (and related problems) will be resolved positively in the near future.

Let us now state the main result of this section.

**Theorem 3.1.** *There exists a $O^\star(2^{O(k \log k)})$-time and polynomial space algorithm for* Edge-Subset-FVS *(which implies an algorithm of the same time complexity for* Subset-FVS*).*

This result resolves the open problem posed by Kawarabayashi and by Saurabh. To achieve this result we use several tools such as iterative compression, the 2-Expansion Lemma, Menger's theorem, Gallai's theorem and the algorithm for the Multiway Cut problem. Some of our ideas were inspired by previous FPT results: the algorithm for Multicut parameterized by $(|\mathcal{T}|, k)$ by Guillemot [86], the $O^\star(37.7^k)$–time algorithm for FVS by Guo et al. [87] and the quadratic kernel for FVS by Thomassé [126].

We do not analyze the value of the exponent in the term in the time complexity that is polynomial in the input size (and hidden in the $O^\star$ notation), as in our algorithm it is far from being linear. The most important reasons for this dependency is that the usage of Gallai's theorem requires finding a maximum matching in an auxiliary graph, and the use of iterative compression gives additional multiplicative factor of $|V|$.

We note that an FPT algorithm for Subset-FVS was independently discovered by Kawarabayashi and Kobayashi [100]. Their algorithm uses significantly different techniques (minor theory) and its dependency on $k$ in the running time is worse than $O^\star(2^{O(k \log k)})$.

**Outline of this section** We start by showing (in Section 3.2.1) that Edge-Subset-FVS is fixed-parameter tractable when parameterized by $|S|$. To obtain fixed-parameter tractability for Edge-Subset-FVS parameterized by $k$, in Section 3.2.2 we develop an algorithm that branches into $O^\star(2^k)$ subinstances with the size of $S$ bounded by $O(k^3)$. Thus, the results of Sections 3.2.1 and 3.2.2 prove that Edge-Subset-FVS is fixed-parameter tractable when parameterized by $k$.

Later, in Section 3.2.3, we improve the algorithm for Edge-Subset-FVS parameterized by $|S|$ so that it runs in $O^\star(2^{O(k \log |S|)})$ time. Our arguments in this section closely follow the approach of Guillemot [86] for Multicut. This

enhanced algorithm allows us to improve the running time of the algorithm for Edge-Subset-FVS to $O^\star(2^{O(k \log k)})$. Finally, for sake of completeness, in Section 3.2.4 we include a reduction from Node Multiway Cut to Subset-FVS in the parameterized setting.

### 3.2.1 A simple algorithm for Edge-Subset-FVS parameterized by $|S|$

In this section we concentrate on solving the Edge-Subset-FVS problem parameterized by $|S|$, which means that our complexity function can be exponentially dependent on the number of edges in the set $S$. This is the first step towards obtaining an FPT algorithm when parameterized by $k$. Observe that we may assume $k < |S|$ since otherwise we may delete one vertex from each edge from the set $S$ thus removing all edges from the set $S$ from our graph.

Let us first introduce some notation. For $G = (V, E)$ denote $G_S = (V, E \setminus S)$. By a *partition* of a set $Z$ we mean such a family $\mathcal{P} = \{P_1, \ldots, P_m\}$, that the $P_i$s are pairwise disjoint and their union is $Z$. We say a partition $\mathcal{P}'$ is a *subpartition* of $\mathcal{P}$ if every element of $\mathcal{P}'$ is contained in some element of $\mathcal{P}$, in this case we call $\mathcal{P}$ a *superpartition* of $\mathcal{P}'$.

In this section we show an FPT algorithm which is easy to understand and later (in Section 3.2.3) we present methods to improve the time complexity. We use the fact that Node Multicut is FPT when parameterized by $(k, |\mathcal{T}|)$ which was shown by Marx [110].

**Theorem 3.2.** *There exists an algorithm solving the* Edge Subset Feedback Vertex Set *problem in $O^\star(f(|S|))$ time, for some computable function $f$.*

*Proof.* Let $T$ be some solution of Edge-Subset-FVS. Our new parametrization, by $|S|$, allows us to guess, by checking all possibilities, the subset $T_S = T \cap V(S)$ that is removed by the solution $T$. Moreover, our algorithm guesses how the set $V(S) \setminus T_S$ is partitioned into connected components in the graph $G_S[V \setminus T]$. Clearly both the number of subsets and of possible partitions are functions of $|S|$. For a partition $\mathcal{P} = \{P_1, \ldots, P_m\}$ of $V(S) \setminus T_S$ we form a multigraph $G_\mathcal{P}$ on the set $\{P_1, \ldots, P_m\}$ by adding an edge $P_iP_j$ for every edge $uv \in S$, where $u \in P_i, v \in P_j$. Now we check whether there exists an edge in $G_\mathcal{P}$ which is not a bridge (in particular, a self-loop at one

vertex in $G_{\mathcal{P}}$ is not a bridge). If that is the case we know that the partition $\mathcal{P}$ does not correspond to any solution of Edge-Subset-FVS, as any simple cycle in $G_{\mathcal{P}}$ can be converted into a simple cycle in $G$ — hence we skip this partition. Otherwise we create a set of pairs $\mathcal{T}$, containing all pairs of vertices from the set $V(S) \setminus T_S$ that belong to different sets in the partition $\mathcal{P}$. Formally $\mathcal{T} = \{(v_i, v_j) : v_i \in P_{i'}, v_j \in P_{j'}, i' \neq j'\}$. Because of the properties of the multigraph $G_{\mathcal{P}}$ it is sufficient to ensure that no pair from the set $\mathcal{T}$ is contained in one connected component, hence in the last step we execute the fixed-parameter algorithm for the Node Multicut problem [110] with parameter $k - |T_S|$. If the call returns a positive answer and a solution $X$, the set $T_S \cup X$ is a solution to Edge-Subset-FVS: the connected components of $G_S[V \setminus (T_S \cup X)]$ induce a partition of $V(S) \setminus (T_S \cup X)$ that is a subpartition of $\mathcal{P}$ and thus all remaining edges of $S$ are bridges in $G[V \setminus (T_S \cup X)]$. In particular, as $G_{\mathcal{P}}$ does not contain any self-loop, for any connected component $C$ of $G_S[V \setminus (T_S \cup X)]$ with vertex set $V_C$ the graph $G[V_C]$ does not contain any edge from $S$. Note that we do not require here that $X \cap V(S) = \emptyset$ nor that the induced partition of $V(S) \setminus (T_S \cup X)$ is exactly the partition $\mathcal{P}$ (being a subpartition is sufficient). On the other hand, if the answer to Edge-Subset-FVS is positive, the Node Multicut call returns a solution for at least one choice of $T_S$ and $\mathcal{P}$, the one implied by the Edge-Subset-FVS solution. Observe that $|\mathcal{T}| = O(|S|^2)$ so the algorithm for Node Multicut runs in $O^\star(f(|S|, k))$ time and we obtain an FPT algorithm for the Edge Subset Feedback Vertex Set problem parameterized by $|S|$. □

**Function** EdgeSubsetFeedbackVertexSet$(G, S, k)$ {parameterized by $(|S|)$}
1: **for all** subsets $T_S \subseteq V(S), |T_S| \leq k$ **do**
2: **for all** partitions $\mathcal{P} = \{P_1, \ldots, P_m\}$ of $V(S) \setminus T_S$ **do**
3: form a multigraph $G_{\mathcal{P}}$ on the set $\{P_1, \ldots, P_m\}$ by adding an edge $P_iP_j$ for every edge $uv \in S$, $u \in P_i$, $v \in P_j$.
4: **if** all edges in $G_{\mathcal{P}}$ are bridges **then**
5: let $\mathcal{T} = \{(v_i, v_j) : v_i \in P_{i'}, v_j \in P_{j'}, i' \neq j'\}$
6: **if** MultiCut$(G_S[V \setminus T_S], \mathcal{T}, k - |T_S|)$ returns $(YES, X)$ **then**
7: **return** $T_S \cup X$
8: **return** $NO$

### 3.2.2 Edge-Subset-FVS parameterized by $k$

In this section we show an FPT algorithm for Edge Subset Feedback Vertex Set.

We begin by noting that, using standard arguments, one can show that Edge-Subset-FVS is *self-reducible* — i.e., if we have an algorithm that solves Edge-Subset-FVS, we can also find a witness: a set $T$ of size at most $k$ that intersects all cycles passing through $S$. The procedure is standard: Assume the answer is positive. For every vertex we check whether it can be a part of a solution by removing it from the graph, decreasing $k$ by one and running our algorithm on the reduced instance. For at least one vertex the answer has to be positive, so we greedily take any such vertex into the solution and proceed inductively.

We now follow the idea of *iterative compression* proposed by Reed et al. [120]. First, note that if $V' \subseteq V$ and $T$ is a feasible solution to an Edge-Subset-FVS instance $(G, S, k)$, then $V' \cap T$ is a feasible solution to the instance $(G[V'], S', k)$, where $S' = S \cap E(G[V'])$. Thus, if the answer for $(G[V'], S', k)$ is negative, so is the answer for $(G, S, k)$. Let $V = \{v_1, v_2, \ldots, v_n\}$ be an arbitrary ordering of the set of vertices of $G$. We consecutively construct solutions to Edge-Subset-FVS for instances $\mathcal{I}_i = (G[V_i], S_i, k)$, where $V_i = \{v_1, v_2, \ldots, v_i\}$ and $S_i = S \cap E(G[V_i])$. When looking for a solution for graph $G[V_{i+1}]$, we use the fact that if $T_i$ is a solution for $\mathcal{I}_i$, then $Z_{i+1} = T_i \cup \{v_{i+1}\}$ is a solution for $(G[V_{i+1}], S_{i+1}, k+1)$ — a solution for our problem with the parameter increased by one.

We start with a standard branching into $2^{|Z_{i+1}|}$ subcases, guessing which vertices from $Z_{i+1}$ are taken into a solution to the instance $\mathcal{I}_{i+1}$. Let us focus on a fixed branch, where we decided to take $T_Z \subseteq Z_{i+1}$ into a solution and denote $Z = Z_{i+1} \setminus T_Z$. We delete $T_Z$ from the graph $G$, reduce $S$ to $S \cap E(G \setminus T_Z)$, and decrease $k$ by $|T_Z|$, arriving at the following subproblem.

Disjoint Edge-Subset-FVS
**Input:** An Edge-Subset-FVS instance $(G, S, k)$ together with a set $Z \subseteq V(G)$ that is a solution to the Edge-Subset-FVS instance $(G, S, |Z|)$.
**Parameter:** $k$ and $|Z|$.
**Question:** Does there exist a solution to $(G, S, k)$ that is disjoint with $Z$?

However, we are not going to provide an algorithm that solves any Disjoint Edge-Subset-FVS instance, but only a *maximal* one. Informally speaking, we are only interested in those of $2^{|Z_{i+1}|}$ branches, where the guessed set $T_Z$ is (inclusion-wise) maximal. Formally:

**Definition 3.3.** We say that a Disjoint Edge-Subset-FVS instance

$(G, S, k, Z)$ is a *maximal* instance if every feasible solution to Edge-Subset-FVS instance $(G, S, k)$ is disjoint with $Z$.

We provide a set of reductions that reduce the size of $S$ to polynomial in $k$. However, we do not require that the reductions are sound with respect to any Disjoint Edge-Subset-FVS instance, but only to the maximal ones. Formally, we define the reductions as follows.

**Definition 3.4.** We say that a Disjoint Edge-Subset-FVS instance $(G', S', k', Z')$ is a *properly reduced* instance $(G, S, k, Z)$ if the following holds:

1. $|V(G')| \leq |V(G)|$ and $k' \leq k$;
2. if $(G, S, k)$ is an Edge-Subset-FVS NO-instance, so is $(G', S', k')$;
3. if $(G, S, k, Z)$ is a maximal YES-instance of Disjoint Edge-Subset-FVS, so is $(G', S', k', Z')$.

We are now ready to state the main theorem of this section.

**Theorem 3.5.** *There exists a polynomial-time algorithm $\mathcal{R}$ that, given a* Disjoint Edge-Subset-FVS *instance $(G, S, k, Z)$, either:*

1. *returns a properly reduced instance $(G', S', k', Z')$ with $k' \leq k$, $|Z'| \leq |Z|$ and $|S'| = O(k'|Z'|^2)$;*
2. *or returns IGNORE, in this case $(G, S, k, Z)$ is not a* maximal Disjoint Edge-Subset-FVS *YES-instance.*

We first show that Theorem 3.5 leads to the desired FPT algorithm for Edge-Subset-FVS.

**Theorem 3.6.** *There exists an algorithm solving the* Edge Subset Feedback Vertex Set *problem in $O^\star(f(k))$ time for some computable function $f$.*

*Proof.* In each step of the iterative compression, in each of $2^{|Z_{i+1}|}$ branches, we run the algorithm $\mathcal{R}$. If it gives the second answer, we ignore this branch. In case of the first answer, we invoke the algorithm from Theorem 3.2 on Edge-Subset-FVS instance $(G', S', k')$, what results in running time $O^\star(f(k))$ for some computable function $f$. Note that if $(G', S', k')$ is an Edge-Subset-FVS YES-instance, so is $(G, S, k)$ (by the second property of Definition 3.4),

and any solution (even not disjoint with $Z$) to $(G, S, k)$ can be extended to a solution of $\mathcal{I}_{i+1}$ by taking its union with $T_Z$. Thus if $\mathcal{I}_{i+1}$ is a NO-instance, the algorithm cannot find a solution. Otherwise, let $T$ be a solution to $\mathcal{I}_{i+1}$ with largest possible intersection with $Z_{i+1}$. We claim that the algorithm finds a solution in the branch $T_Z = T \cap Z_{i+1}$. Indeed, then $(G, S, k, Z)$ is a maximal YES-instance to Disjoint Edge-Subset-FVS and the algorithm $\mathcal{R}$ cannot return IGNORE. Thus we obtain a Edge-Subset-FVS YES-instance $(G', S', k')$, and the algorithm from Theorem 3.2 finds a solution. □

In Section 3.2.3 we develop a faster version of the algorithm from Theorem 3.2 so that the running time of the algorithm for Edge-Subset-FVS becomes $O^\star(2^{k \log k})$.

The proof of Theorem 3.5 consists of a set of polynomial-time *proper reductions* (in the sense of Definition 3.4), each either decreasing $|V(G)|$ or decreasing $|E(G)|$ while not changing $|V(G)|$. Some reductions may result with an IGNORE answer, in which case the answer is immediately returned from this branch. Note that in this case the last property of Definition 3.4 implies that all Disjoint Edge-Subset-FVS instances in the current sequence of reductions are not maximal YES-instances. We assume that at each step, the lowest–numbered applicable reduction is used. If no reduction is applicable, we claim that $|S| = O(k|Z|^2)$.

We start with an obvious reduction. Note that if it is not applicable, every edge in $S$ is contained in some simple cycle.

**Reduction 1.** Remove all bridges and all connected components not containing any edge from $S$.

### The outer–abundant lemma

In this section we consider an instance of Disjoint Edge-Subset-FVS $(G, S, k, Z)$, where $G = (V, E)$. We assume that Reduction 1 is not applicable, i.e., every edge in $S$ belongs to some simple cycle. The approach here is based on ideas from the quadratic kernel for the classical Feedback Vertex Set problem [126], however, a few aspects need to be adjusted to better fit our needs.

**Definition 3.7.** A set $F \subseteq V$ is called *outer–abundant* when:

(a) $G[F]$ is connected,

(b) there are no edges from $S$ in $G[F]$,

(c) there at least $10k$ edges from $S$ incident with $F$.

**Lemma 3.8** (The outer–abundant lemma)**.** *Let $F$ be an outer–abundant set. If Reduction 1 is not applicable, then in polynomial time one can either:*

- *find a nonempty set $X \subseteq V \setminus F$ such that the following condition is satisfied: if there exists a solution $A$ for* Edge-Subset-FVS *on $(G, S, k)$ such that $A \cap F = \emptyset$, then there exists a solution $A'$ such that $A' \cap F = \emptyset$ and $X \subseteq A'$;*
- *or correctly state that any solution for* Edge-Subset-FVS *on $(G, S, k)$ is not disjoint with $F$.*

Before we start proving Lemma 3.8, let us recall a few tools used in the quadratic kernel for Feedback Vertex Set [126]. First, we recall the result of Gallai on finding disjoint $A$–paths.

**Theorem 3.9** (Gallai [81])**.** *Let $A$ be a subset of vertices of a graph $G$. A path is called an $A$–path if its endpoints are different vertices in $A$. If the maximum number of vertex disjoint $A$–paths is at most $k$, there exists a set of vertices $B' \subseteq V$ of size at most $2k$ intersecting every $A$–path.*

Moreover, it follows from Schrijver's proof of the Gallai's theorem [123] that Theorem 3.9 can be algorithmized: in polynomial time we can find either $k+1$ disjoint $A$–paths or the set $B'$.

The other theorem we need is the 2–Expansion Lemma (see Figure 3.1 for an illustration):

**Theorem 3.10** (2–Expansion Lemma, Theorem 2.3 in [126])**.** *Let $H$ be a nonempty bipartite graph on bipartition $(X, Y)$ with $|Y| \geq 2|X|$ and such that every vertex of $Y$ has at least one neighbour. Then there exist nonempty subsets $X' \subseteq X$, $Y' \subseteq Y$ such that $N(Y') \cap X = X'$ and one can assign to each $x \in X'$ two* private *neighbours $y_1^x, y_2^x \in Y'$ (i.e., each $y \in Y'$ is assigned to at most one $x \in X'$). In addition, such pair of subsets $X'$, $Y'$ can be computed in polynomial time in the size of $H$.*

*Proof of Lemma 3.8.* An $S$–cycle is called *important* if it contains an edge from $S \cap E(F, V \setminus F)$. A set of important cycles $\{C_1, C_2, \ldots, C_t\}$ such that the sets of vertices $C_i \setminus F$ are pairwise disjoint is called a *$t$–flower*.

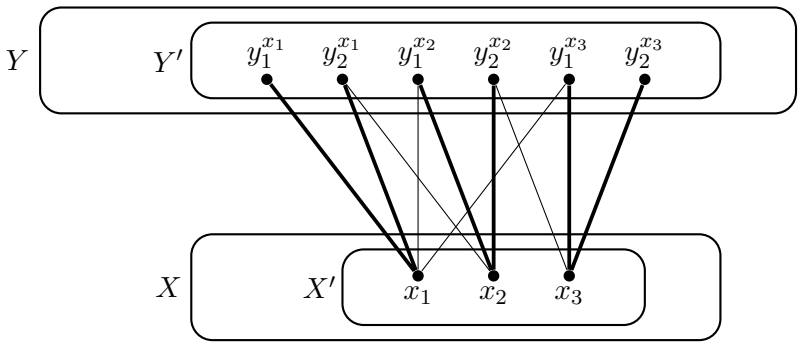


Figure 3.1: Illustration of the 2–Expansion Lemma (Theorem 3.10).

Note that if there exists a vertex $v \in V \setminus F$ such that $|E(\{v\}, F)| \geq 2$ and $E(\{v\}, F) \cap S \neq \emptyset$, then one can take $X = \{v\}$. Indeed, any solution disjoint with $F$ has to include $v$, since by connectivity of $G[F]$ there is an important cycle contained in $G[F \cup \{v\}]$ passing through $v$ via at least one edge from $S$. Thus we can assume that each vertex from $V \setminus F$ is connected to $F$ by a number of edges (possibly zero) not belonging to $S$ or by a single edge from $S$.

Now we prove that in polynomial time we can find one of the following structures: either a $(k+1)$–flower, or a set $B$ of at most $3k$ vertices belonging to $V \setminus F$ such that each important cycle passes through at least one of them (further called a $3k$–*blocker*).

Let $\mathcal{C}$ be the set of those important cycles, which contain exactly two edges between $F$ and $V \setminus F$ (intuitively, visiting $F$ only once). Note that due to connectivity of $G[F]$, a set is a $3k$–blocker iff any cycle from $\mathcal{C}$ passes through at least one of its elements. For $C \in \mathcal{C}$ let us examine its two edges between $F$ and $V \setminus F$. As $C$ is important, one or two of them belong to $S$. We say that such a cycle is of type I iff exactly one of these edges belong to $S$ and of type II otherwise.

Firstly, we sort out the type I cycles. We remove $F$ from the graph and replace it with two vertices $s$ and $t$. Also we add edges incident with $s$ and $t$ — for every $vw \in E(F, V \setminus F)$ with $v \in F$, we add edge $sw$ if $vw \in S$ and edge $wt$ otherwise. By a simple application of the vertex max–flow algorithm and Menger's theorem one obtains either a vertex–disjoint set of paths between $s$ and $t$ of cardinality $k+1$, or a set of at most $k$ vertices such that each such a path passes through at least one of them. Returning to the original graph

transforms each path between $s$ and $t$ into a type I cycle, so we have found either a $(k+1)$-flower or a $k$-blocker of type I cycles.

Now, we deal with the type II cycles. Let $J \subseteq V \setminus F$ be the set of vertices that are connected to $F$ by an edge from $S$. We remove temporarily $F$ from the graph and apply Theorem 3.9 to the set $J$. Note that a set of $k+1$ vertex–disjoint $J$–paths corresponds to a $(k+1)$–flower, and the set $B'$ is a $2k$–blocker of type II cycles.

Using both of these methods we obtain either a $(k+1)$–flower or, by taking a union of blockers, a $3k$–blocker. Note that both algorithms run in polynomial time.

The existence of a $(k+1)$–flower immediately shows that a solution $A$ disjoint with $F$ does not exist, as each cycle belonging to a flower has to include at least one vertex from $A$. Thus we are left only with the case of a $3k$-blocker. Let us denote it by $B$.

Let us examine $G[V \setminus (F \cup B)]$. Let $H = (V_H, E_H)$ be any of its connected components. Note that $H$ is connected to $F$ by a number of edges (possibly zero) not belonging to $S$ or by a single edge from $S$. Indeed, otherwise, due to connectedness of $H$ and of $G[F]$, there would be an important cycle contained in $G[F \cup V_H]$ not blocked by $B$. Using observation from the second paragraph of this proof, we may assume that each vertex from $B$ is connected to $F$ by at most one edge from $S$. So there are at most $3k$ edges from $S$ between $B$ and $F$. As there are at least $10k$ edges from $S$ between $F$ and $V \setminus F$, we have at least $7k$ connected components of $G[V \setminus (F \cup B)]$ connected to $F$ by a single edge from $S$.

We call a component $H$ *easy* if there is an $S$–cycle fully contained in $H$. If the number of easy components is larger than $k$ (what can be verified in polynomial time), there is more than $k$ vertex–disjoint $S$–cycles, so $A$ does not exist. Thus we may assume that there are at least $6k$ non–easy components connected to $F$ by a single edge from $S$. We call them *tough* components.

Let $H = (V_H, E_H)$ be a tough component. Observe that $N(V_H) \cap B \neq \emptyset$. Indeed, otherwise the only edge between $V_H$ and $V \setminus V_H$ would be the edge from $S$ connecting $H$ with $F$ and thus a bridge sorted out by Reduction 1.

Let $T$ be the set of tough components. We construct a bipartite graph $(B \cup T, E_{exp})$ such that $vH \in E_{exp}$ for $v \in B$, $H \in T'$ iff $v \in N(V_H)$. Note that due to observation in the previous paragraph, $(B \cup T, E_{exp})$ satisfy assumptions of Theorem 3.10, as $|T| \geq 6k = 2 \cdot 3k \geq 2|B|$. So we have nonempty sets $X \subseteq B$ and $Y \subseteq T$ such that for every $v \in X$ there are two

*private* tough components $H_{v,1}, H_{v,2} \in Y$ with $v \in N(V_{H_{v,i}})$ for $i = 1, 2$ and $B \cap \bigcup_{H \in Y} N(H) = X$ (see Figure 3.2).

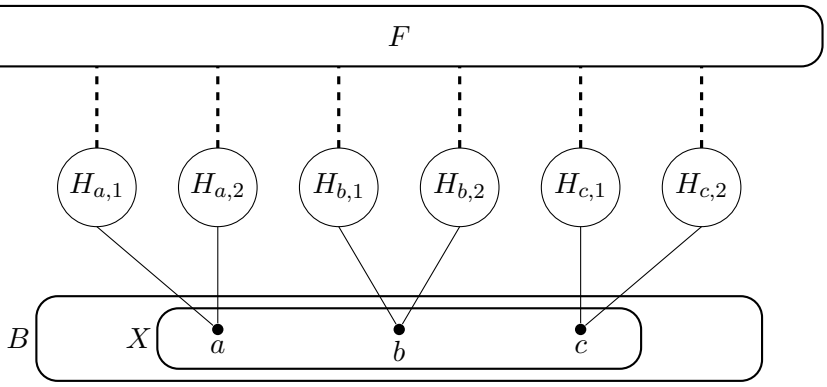


Figure 3.2: Illustration of the application of the 2–Expansion Lemma to the proof of Lemma 3.8. Dashed edges belong to the set $S$.

Let $v \in X$. Note that due to the connectedness of $H_{v,i}$ and $G[F]$, there is an $S$–cycle $C_v$ passing through $v$ — it goes from $v$ to $H_{v,1}$, then to $F$ through an edge from $S$, then to $H_{v,2}$ through an edge from $S$ and back to $v$. Assume that $A$ is a solution to the Edge-Subset-FVS on $(G, S, k)$ and $A \cap F = \emptyset$. We see that cycles $C_v$ for $v \in X$ form an $|X|$–flower, so there are at least $|X|$ vertices in $A \cap (X \cup \bigcup_{H \in Y} V_H)$. On the other hand, each $S$–cycle passing through any vertex from $X \cup \bigcup_{H \in Y} V_H$ passes through a vertex from $X$. Indeed, each $H \in Y$ is connected to $V \setminus V_H$ with a single edge incident with $F$ and a number of edges incident with $X$. Hence each cycle passing through a vertex from $\bigcup_{H \in Y} V_H$ is fully contained in some $H \in Y$ or goes from some $H \in Y$ to $X$. As each $H \in Y$ is non–easy, cycles not incident with $X$ do not contain any edge from $S$.

These observations prove that if we construct

$$A' = \left( A \setminus \bigcup_{H \in Y} V_H \right) \cup X,$$

$A'$ will be still a solution to Edge-Subset-FVS. Thus the set $X$ satisfies all the required conditions. □

Lemma 3.8 allows us to greedily assume $X$ is in the solution we are looking for (after we ensure that it is disjoint with $F$), and either take it into the

solution (if $X \cap Z = \emptyset$) or return IGNORE (if $X \cap Z \neq \emptyset$). As a direct application of Lemma 3.8, we obtain the following reduction rule. Note that if it is not applicable, there are at most $10k|Z|$ edges from $S$ incident with $Z$.

**Reduction 2.** Let $v \in Z$ be a vertex that is incident to at least $10k$ edges from $S$. Apply Lemma 3.8 to the outer–abundant set $F = \{v\}$, If a set $X$ is returned and $X \cap Z = \emptyset$, we remove $X$ and decrease $k$ by $|X|$, otherwise we return IGNORE.

### Bubbles

Recall that our goal is to reduce the size of $S$. After Reduction 2, there are $O(k|Z|)$ edges from $S$ incident with $Z$. Thus, we need to care only about $S \cap E(G[V \setminus Z])$.

As $Z$ is a feasible solution to Edge-Subset-FVS on $(G, S, |Z|)$, every edge from $S \cap E(G[V \setminus Z])$ has to be a bridge in $G[V \setminus Z]$. After removing those bridges $G[V \setminus Z]$ becomes a union of connected components not having any edge from $S$. We call each such a component a *bubble*. Denote the set of bubbles by $\mathcal{D}$. On $\mathcal{D}$ we have a natural structure of a graph $H = (\mathcal{D}, E_{\mathcal{D}})$, where $IJ \in E_{\mathcal{D}}$ iff components $I$ and $J$ are connected by an edge from $S$. As $Z$ is a solution, $H$ is a forest and each $I, J$ connected in $H$ are connected in $G$ by a single edge from $S$.

Consider $I \in \mathcal{D}$. Denote the set of vertices of $I$ by $V_I$. Note that if at most one edge leaves $I$ (that is, $|E(V_I, V \setminus V_I)| \leq 1$) then $V_I$ would be removed while processing Reduction 1. The following reduction sorts out bubbles with exactly two outgoing edges and later we assume that for every $I \in \mathcal{D}$ we have $|E(V_I, V \setminus V_I)| \geq 3$.

**Reduction 3.** Let us assume that $|E(V_I, V \setminus V_I)| = 2$ and let $\{u, v\} = N(V_I)$ (possibly $u = v$). Each cycle passing through a vertex from $V_I$ is either fully contained in $I$ and thus non–$S$–cycle, or exits $V_I$ through $u$ and $v$. We remove $V_I$ from the graph and replace it with a single edge $uv$, belonging to $S$ iff any one of the two edges in $E(V_I, V \setminus V_I)$ is in $S$.

If the addition of the edge $uv$ lead to a multiple edge or a loop, we immediately resolve it:

- If $uv$ is a loop and $uv \notin S$, we delete it.

- If $uv$ is a loop and $uv \in S$, we return IGNORE, as the fact that $Z$ is a solution to $(G, S, |Z|)$ implies that $u \in Z$.
- If $uv$ is a multiple edge and no edge between $u$ and $v$ is from $S$, we delete the new edge $uv$.
- If $uv$ is a multiple edge and one of the edges between $u$ and $v$ is $S$, we first note that, since $Z$ is a solution to $(G, S, |Z|)$, $u$ or $v$ is in $Z$. If both are in $Z$, we return IGNORE, otherwise we delete $\{u, v\} \setminus Z$ from the graph and decrease $k$ by one.

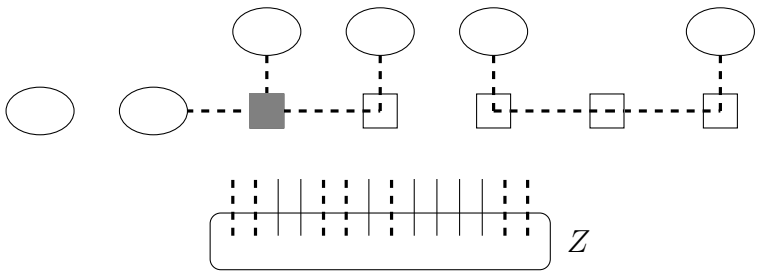


Figure 3.3: Set of vertices $Z$ and a forest of bubbles. Ellipse-shaped bubbles represent leaf bubbles, white squares represent edge bubbles and squares filled with gray represent inner bubbles. Dashed edges belong to the set $S$.

We are now left with bubbles that have at least three outgoing edges. We classify those bubbles according to the number of edges that connect them to other bubbles, that is $\deg_H(I)$.

**Definition 3.11.** We say that a bubble $I \in \mathcal{D}$ is

(a) *a solitary bubble* if $\deg_H(I) = 0$,

(b) *a leaf bubble* if $\deg_H(I) = 1$,

(c) *an edge bubble* if $\deg_H(I) = 2$,

(d) *an inner bubble* if $\deg_H(I) \geq 3$.

Denote by $\mathcal{D}_s, \mathcal{D}_l, \mathcal{D}_e, \mathcal{D}_i$ the sets of appropriate types of bubbles.

See also Figure 3.3 for an illustration.

We show that we can do some reductions to make following inequalities hold:

$$|\mathcal{D}_l| = O(k|Z|^2), \qquad |\mathcal{D}_i| < |\mathcal{D}_l|, \qquad |\mathcal{D}_e| < 3(|Z| + k) + |\mathcal{D}_i| + |\mathcal{D}_l|.$$

Note that these conditions imply that $|\mathcal{D} \setminus \mathcal{D}_s| = O(k|Z|^2)$. As edges of $H$ create a forest over $\mathcal{D} \setminus \mathcal{D}_s$, this bounds the number of edges from $S$ not incident with $Z$ by $O(k|Z|^2)$, as desired.

**Lemma 3.12.** $|\mathcal{D}_i| < |\mathcal{D}_l|$.

*Proof.* As $H$ is a forest, then $|E_{\mathcal{D}}| < |\mathcal{D}| - |\mathcal{D}_s| = |\mathcal{D}_i| + |\mathcal{D}_e| + |\mathcal{D}_l|$. Moreover, $2|E_{\mathcal{D}}| = \sum_{I \in \mathcal{D}} \deg_H(I) \geq 3|\mathcal{D}_i| + 2|\mathcal{D}_e| + |\mathcal{D}_l|$. Therefore $|\mathcal{D}_l| > |\mathcal{D}_i|$. □

**Reduction 4.** If $|\mathcal{D}_e| \geq 3(|Z| + k) + |\mathcal{D}_i| + |\mathcal{D}_l|$, then return IGNORE.

**Lemma 3.13.** *Reduction 4 is a proper reduction.*

*Proof.* We first show that the number of edge bubbles not adjacent to any other edge bubble in $H$ is at most $|\mathcal{D}_i| + |\mathcal{D}_l|$. Let us root each connected component of $H$ in an arbitrary leaf and for any bubble $I \in \mathcal{D}_e$ let $\varphi(I)$ be the only child of $I$. Observe that this mapping is injective and maps the set of edge bubbles isolated in $H[\mathcal{D}_e]$ into $\mathcal{D}_i \cup \mathcal{D}_l$.

We now prove by contradiction that if the reduction is applicable, then every feasible solution of $(G, S, k)$ contains a vertex from $Z$. As edge bubbles have degree 2 in $H$, $H[\mathcal{D}_e]$ is a set of paths of non-zero length and isolated vertices. There are at least $3(|Z| + k)$ vertices contained in the paths. Let $M$ be a maximum matching in $H[\mathcal{D}_e]$. If a path contains $l$ vertices (for $l \geq 2$), it has a matching of cardinality $\lfloor \frac{l}{2} \rfloor \geq \frac{l}{3}$. Therefore, $|M| \geq |Z| + k$. Let us examine an arbitrary $IJ \in M$. Recall that at least three edges leave each bubble. As each of $V_I$, $V_J$ is adjacent to two other bubbles by single edges, it has to be connected to $Z$ as well. Choose $u_I, u_J$ — vertices from $Z$ such that $u_I \in N(V_I)$ and $u_J \in N(V_J)$ (possibly $u_I = u_J$). We see that there is a path from $u_I$ to $u_J$ passing through an edge from $S$: it goes from $u_I$ to $I$, then to $J$ through an edge from $S$, and then to $u_J$. As $M$ is a matching, such paths are vertex–disjoint for all $IJ \in M$, except for endpoints $u_I$ and $u_J$.

Recall $|M| \geq |Z| + k$. Hence, if we have a solution disjoint with $Z$ of cardinality at most $k$, there are at least $|Z|$ pairs $IJ \in M$, where neither $I$ nor $J$ contains a vertex from the solution. Now we construct a graph

$P = (Z, E_P)$ such that $u_I u_J \in E_P$ if $u_I, u_J$ have been chosen for some $IJ \in M$, where $I$ and $J$ are solution–free. We prove that $P$ has to be a forest. Indeed, otherwise there would be a cycle in $P$ — and by replacing each edge from it by associated path, we construct an $S$-cycle in $G$ (as paths from which edges from $E_P$ originated are vertex–disjoint). This cycle does not contain any vertex from the solution, as it passes only through $Z$ and solution–free bubbles. Hence, $P$ is indeed a forest. However, as $|E_P| \geq |Z|$, $P$ cannot be a forest; the contradiction ends the proof. □

### The leaf bubble reduction

We are left with the leaf bubbles and we need to show reductions that lead to $|\mathcal{D}_l| = O(k|Z|^2)$. We do this by a single large reduction described in this subsection. It proceeds in a number of steps. Each step either returns IGNORE (thus ending the reduction) or — after, possibly, modifying $G$ — passes to the next step. Each step is not a standalone reduction, as it may increase $|E(G)|$. However, if the reduction below is fully applied, it either returns IGNORE or reduces $|V(G)|$.

Let $I$ be a leaf bubble. As there are at least three edges leaving $I$, each leaf bubble is connected to $Z$ by at least two edges. We begin with a bit of preprocessing:

**Step 1.** As long as there are two vertices $v, v'$ in $Z$ with $vv' \notin E$, and at least $k+1$ bubbles, each connected to both $v$ and $v'$ by edges not in $S$, we add an edge $vv'$ to $E$ (but not to $S$).

**Lemma 3.14.** *The output $(G', S, k)$ of Step 1 and the input $(G, S, k)$, as* Edge-Subset-FVS *instances, have equal sets of feasible solutions.*

*Proof.* Obviously any solution to $(G', S, k)$ is a solution to $(G, S, k)$, as we only added edges (and thus only added potential $S$–cycles). On the other hand, suppose we have a solution $T$ to $(G, S, k)$ which is not a solution to $(G', S, k)$. Then there is some $S$–cycle $C$ in $(G', S, k)$ not passing through any vertex of $T$. $C$ has to pass through the edge $vv'$ (otherwise it would also be an $S$–cycle in $(G, S, k)$).

As $|T| \leq k$, among the bubbles required by Step 1 there is at least one bubble $I_j$ which is disjoint with $T$. Thus we can find a simple path $P$ connecting $v$ and $v'$, the interior vertices of which are all in $I_j$. Note that as $I_j$ is a bubble and the edges to $v$ and $v'$ were not in $S$, $P$ does not contain any

edge from $S$. Consider the cycle $C'$ (not necessarily simple) in $G$ which is formed by replacing the edge $vv'$ in $C$ by the path $P$. As $C$ was an $S$–cycle, there is some edge $e \in S \cap C$. This edge is visited by $C'$ exactly once — as we took out $vv' \notin S$ and added edges from $P$, which is disjoint with $S$. If we consider the graph spanned by edges from $C'$, the edge $e$ is not a bridge in this graph, as the endpoints of $e$ are connected by $C' \setminus \{e\}$. Therefore, $e$ lies on some simple cycle contained in $C'$, a contradiction. □

Now for each bubble $I$ with vertex set $V_I$ we choose arbitrarily two of the edges connecting it to $Z$: $e_I$ and $e'_I$. Additionally assume that if $S \cap E(V_I, Z) \neq \emptyset$ then $e_I \in S$. Let $v_I$ and $v'_I$ be the endpoints of $e_I$ and $e'_I$ in $Z$ (possibly $v_I = v'_I$). We say that a bubble $I$ is *associated* with vertices $v_I, v'_I$. If two leaf bubbles $I_1, I_2$ are connected in $H$ (form a $K_2$ in $H$), by an edge $e_{I_1 I_2} \in S$, we call them a *bubble–bar*. The proofs of the following lemmata proceed along lines similar to the proof of correctness for Reduction 4 (see Figure 3.4 for an illustration):

**Lemma 3.15.** *If there are at least $|Z|^2(k+2)$ leaf bubbles $I$ such that $e_I \in S$ then every feasible solution of* Edge-Subset-FVS *on $(G, S, k)$ contains a vertex from $Z$.*

*Proof.* By the Pigeonhole Principle, there exist $v, v' \in Z$ associated with at least $k+2$ of the considered leaf bubbles. If $v = v'$, there are $k+2$ $S$–cycles sharing only $v$, each constructed from a different bubble $I$ by closing a path contained in $I$ with edges $e_I$ and $e'_I$. Therefore, $v$ needs to be part of any feasible solution. If $v \neq v'$, for each of the $k+2$ bubbles one can similarly choose a path between $v$ and $v'$ which contains an edge from $S$. These $k+2$ paths are vertex–disjoint apart from $v$ and $v'$, so any feasible solution disjoint with $Z$ leaves at least two of them solution–free. These two paths can be arranged into an $S$–cycle, so any feasible solution contains $v$ or $v'$. □

**Lemma 3.16.** *If there are at least $|Z|^2(k+1)$ leaf bubbles $I$ such that $v_I v'_I \in S$, then every feasible solution of* Edge-Subset-FVS *on $(G, S, k)$ contains a vertex from $Z$.*

*Proof.* As before, there exist $v, v' \in Z$ associated with at least $k+1$ considered leaf bubbles. These bubbles generate at least $k+1$ $S$–cycles, which are vertex–disjoint apart from $v, v'$. Therefore, any feasible solution needs to include $v$ or $v'$. □

**Lemma 3.17.** *If there are at least $|Z|^2(k+2)$ bubble–bars, then any feasible solution of* Edge-Subset-FVS *on $(G, S, k)$ contains a vertex from $Z$.*

*Proof.* By the Pigeonhole Principle, there exist $v, v' \in Z$ such that there exist at least $k+2$ bubble–bars $(I_1, I_2)$ with $v_{I_1} = v$ and $v_{I_2} = v'$. If $v = v'$, there are $k+2$ $S$–cycles having only $v$ in common (one through each bubble–bar), so any feasible solution has to contain $v$. If $v \neq v'$, there are $k+2$ paths connecting $v$ and $v'$ and sharing only $v$ and $v'$. Any solution disjoint with $Z$ would leave at least two of them solution–free. Then these two paths could be arranged into a solution–free $S$–cycle. □

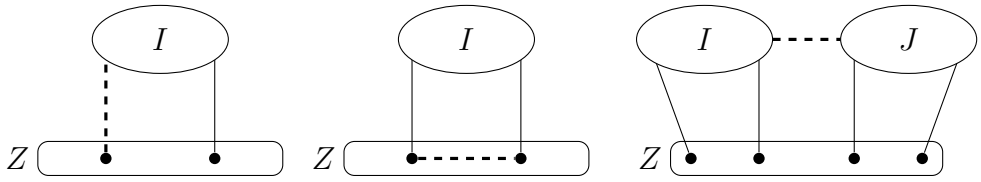


Figure 3.4: Situations that trigger Lemmata 3.15, 3.16 and 3.17. Dashed edges belong to the set $S$.

The above lemmata justify our next step.

**Step 2.** If any of the situations from Lemmata 3.15, 3.16 and 3.17 occur, return IGNORE.

Summing all the obtained bounds, we can count almost all the leaf bubbles (possibly more than once) and bound their number by $O(k|Z|^2)$. In the end of this section we will show that the ones that are left satisfy the following definition (see Figure 3.5 for an illustration):

**Definition 3.18.** A leaf bubble $I$ satisfying the following three conditions is called a *clique bubble*:

(a) $G[N(V_I) \cap Z]$ is a clique not containing any edge from $S$,

(b) $I$ is connected to $Z$ by edges not belonging to $S$,

(c) $I$ is connected to a non–leaf bubble.

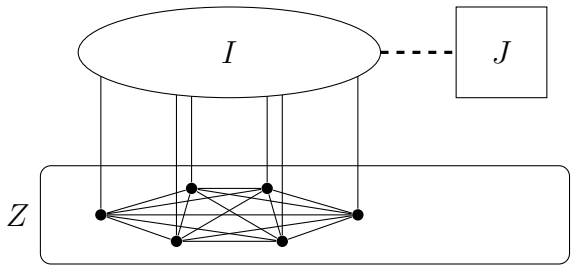


Figure 3.5: A clique bubble $I$. The dashed edge belongs to $S$. The bubble $J$ is and an edge bubble or an inner bubble.

Denote the only edge from $S$ connecting a given clique bubble $I$ with its neighbour bubble by $w_I w'_I$, with $w_I \in V_I$.

**Lemma 3.19.** *If there exists a feasible solution $T$ for* Edge-Subset-FVS *on $(G, S, k)$, then there exists a feasible solution $T'$, which is disjoint from all clique bubbles in $G$. Moreover, if $T$ is disjoint with $Z$, so is $T'$.*

*Proof.* Let $I$ be a clique bubble. Assume we have a feasible solution $T$, with $T \cap V_I \neq \emptyset$. We show $T' = (T \setminus V_I) \cup \{w'_I\}$ is also a feasible solution. Consider any $S$–cycle $C$ in $G[V \setminus T']$. This cycle has to pass through $V_I$ (possibly multiple times), or it would be an $S$–cycle in $G[V \setminus T]$, contrary to the assumption that $T$ was a feasible solution. Note that $C$ has to enter and exit $V_I$ through $N(V_I) \cap Z$, as the only vertex in $N(V_I) \setminus Z$ is $w'_I$, which is removed by $T'$. But then $C$ can be shortened to $C'$ by replacing every part contained in $V_I$ by a single edge in $Z$ (as $N(V_I) \cap Z$ is a clique). Now $C'$ is disjoint from $V_I$ and is an $S$–cycle due to the definition of the clique bubble. So $C'$ is an $S$–cycle in $G[V \setminus T]$, a contradiction.

Note that the only vertex we added to $T$ was $w'_I$, which does not belong to a clique bubble (it does not belong even to a leaf bubble, from property (c) in the definition of clique bubbles). Thus we can apply this procedure inductively, at each step reducing the number of vertices in $T$ contained in clique bubbles, until none are left. □

Assume there is a vertex $v \in Z$ such that $v \in N(V_{I_j})$ for some distinct clique–bubbles $I_1, I_2, \ldots, I_{10k}$. We show that the set $F = \{v\} \cup \bigcup_{j=1}^{10k} V_{I_j}$ is outer–abundant in $G$. Indeed, it is connected and due to the definition of bubbles and properties of the clique bubble definition, the subgraph $G[F]$

does not contain edges from $S$. Moreover, there are at least $10k$ edges from $S$ incident with $G[F]$ — these are the edges connecting bubbles $I_j$ with other bubbles, not contained in $F$ as they are non–leaf ones due to property (c). This enables us to formulate the key step:

**Step 3.** If there is a vertex $v \in Z$ which is adjacent to at least $10k$ clique bubbles, we apply Lemma 3.8 to the set $F = \{v\} \cup \bigcup_{j=1}^{10k} V_{I_j}$. If a set $X$ is returned and $X \cap Z = \emptyset$, we remove $X$ from the graph and decrease $k$ by $|X|$, otherwise we return IGNORE.

Suppose there is a feasible solution $T$ to $(G, S, k)$. Due to Lemma 3.19 we may assume $T$ is disjoint with $F \setminus \{v\}$. Thus either $T$ contains $v$, or it is disjoint with $F$, and by Lemma 3.8 there exists a solution containing $X$. This justifies the correctness of Step 3.

Now we summarize the steps made in this section to show clearly that the number of leaf bubbles is bounded by $O(k|Z|^2)$.

Assume no reduction is applicable. Note that in the last run, the last reduction may add some edges in Step 1. Let $G'$ denote the modified graph. Let us check that the graph $G'$ indeed has $O(k|Z|^2)$ edges from $S$:

1. The decomposition of $V(G) \setminus Z$ into bubbles is the same as the decomposition of $V(G') \setminus Z$ and bubbles that were inner or edge bubbles in $G$ are, respectively, inner or edge bubbles in $G'$;

2. If Step 2 is not applicable, there are at most $|Z|^2(k+2)-1$ leaf bubbles connected to $Z$ by an edge from $S$ by Lemma 3.15, at most $|Z|^2(k+1)-1$ leaf bubbles associated with a pair of vertices connected with an edge from $S$ by Lemma 3.16, and at most $2|Z|^2(k+2)$ leaf bubbles connected to other leaf bubbles by Lemma 3.17.

3. If Step 1 is not applicable, for any pair $v, v'$ of vertices in $Z$ with $vv' \notin E$ there are at most $k$ leaf bubbles adjacent to both vertices of that pair through edges not in $S$.

4. If a leaf bubble is not a clique bubble, it either has an edge from $S$ between some two of its neighbours in $Z$ (property (a)), has some two neighbours in $Z$ not connected by an edge (property (a)), is connected to $Z$ by an edge in $S$ (property (b)), or is connected to a leaf bubble, forming a bubble–bar (property (c)). The number of such bubbles in all four cases was estimated above. Thus, in total, there are at most $O(k|Z|^2)$ bubbles which are not clique bubbles.

5. Finally, if Step 3 is not applicable, there are at most $(10k-1)|Z|$ clique bubbles.

6. Thus $|\mathcal{D}_l| = O(k|Z|^2)$, moreover $|\mathcal{D}_i| \leq |\mathcal{D}_l|$ by Lemma 3.12 and $|\mathcal{D}_e| \leq 3(|Z| + k) + |\mathcal{D}_i| + |\mathcal{D}_l|$ by Reduction 4 — thus the number of edges in $S$ not incident with $Z$ is bounded by $O(k|Z|^2)$. We added no new edges to $S$, and, since Reduction 2 was not applicable, the number of edges in $S$ incident to $Z$ was bounded by $O(k|Z|)$ in the input graph, thus in the output graph there are $O(k|Z|^2)$ edges from $S$, as desired.

Thus we managed to reach the state when the number of leaf bubbles is bounded by $O(k|Z|^2)$. As we modified only the subgraph $G[Z]$, the sets $\mathcal{D}_i$, $\mathcal{D}_e$, $\mathcal{D}_l$ remain the same after modifications and we obtain a graph with $|S| = O(k|Z|^2)$. Hence the proof of Theorem 3.5 is finished.

### 3.2.3 Improving the time complexity of Edge-Subset-FVS parameterized by $|S|$

The goal of this section is to improve the time complexity of the algorithm for Edge Subset Feedback Vertex Set parameterized by $|S|$ (Theorem 3.2). Our main result in this section is the following:

**Theorem 3.20.** *There exists an algorithm solving the* Edge Subset Feedback Vertex Set *problem in* $O^\star(2^{O(k \log |S|)})$ *time.*

We first prove Theorem 3.1 using Theorem 3.20.

*Proof of Theorem 3.1.* We proceed as in the proof of Theorem 3.6, but we use the algorithm from Theorem 3.20 instead of the algorithm from Theorem 3.2. Recall that in the proof of Theorem 3.6, in each step of the iterative compression, in each of the $O(2^k)$ branches, we reduce the instance using Theorem 3.5 so that $|S| = O(k^3)$. We note that on such instances the algorithm from Theorem 3.20 runs in $O^\star(2^{k \log k})$ time. This concludes the proof of Theorem 3.1. □

Let us now proceed to the proof of Theorem 3.20. Our whole approach is closely based on the arguments of Guillemot [86] for Multicut. We first recall that Node Multiway Cut is fixed-parameter tractable when parameterized by the solution size $k$, and currently the fastest algorithm runs in $O^\star(2^k)$ time and polynomial space [46].

*Proof of Theorem 3.20.* Assume we are given a Edge Subset Feedback Vertex Set instance $(G, S, k)$, where $G = (V, E)$. The algorithm works in three phases. In the first two phases we aim to partition the set of all endpoints of edges from $S$ into a family of subsets. More formally, the first two phases generate a set of pairs of the form $(\mathcal{P}, R)$ (called *partition pairs*), where:

1. $R \subseteq V$ and $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ is a partition of $V(S) \setminus R$ (i.e., $P_i \cap P_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{m} P_i = V(S) \setminus R$);

2. for any set $T \subseteq V$ of size at most $k$ there exists a generated partition pair $(\mathcal{P}, R)$ such that $T \cap V(S) \subseteq R \subseteq T$ and the partition that the connected components of $G_S[V \setminus T]$ induce on $V(S) \setminus R$ is exactly $\mathcal{P}$.

In the third phase, for each generated partition pair $(\mathcal{P}, R)$ we check whether it is possible to remove a set $T \supseteq R$ of at most $k$ vertices so that the partition induced by the connected components of $G_S[V \setminus T]$ on $V(S) \setminus R$ equals $\mathcal{P}$ and whether the generated partition pair $(\mathcal{P}, R)$ implies that we removed all cycles passing through $S$ from $G$. In what follows we first describe the three-phase algorithm in detail and then we show its correctness.

Initialize $R = \emptyset$. The first phase works as follows:

1. Select a spanning forest $F$ of $G_S[V \setminus R]$, let $U = V(S) \setminus R$ be the set of endpoints of edges from $S$ outside $R$;

2. If $k = |R|$ proceed directly to phase two;

3. As long as $F$ contains isolated vertices not from $U$ or leaves not from $U$ in $F$, remove them from $F$;

4. As long as $F$ contains vertices of degree 2 not from $U$ in $F$, remove them from $F$, and connect the two neighbours of the removed vertex with an edge in $F$;

5. Branch out — one branch passes the resulting forest $F$ and sets $U$ and $R$ to the second phase. In other $|F|$ branches we select a vertex from $F$, add it to $R$ and go back to Point 1.

Note that after the first four steps of phase one $F$ has at most $2|V(S)|$ vertices (as all its vertices of degree at most 2 are from $U \subseteq V(S)$). Thus in the fifth step we have at most $2|V(S)| + 1$ branches. As we can branch out

into phase one at most $k$ times due to steps 2 and 5, this assures we have at most $(2|V(S)|+1)^k$ entries into phase two from phase one. The internal workings of the first phase are obviously polynomial–time.

The second phase is somewhat more complicated, and aims at arriving at a partition of the set $U = V(S)\setminus R$, based on the forest $F$ received from phase one. Informally speaking, the set $R$ is to be included in the solution and in the second phase we choose a subgraph of $F$ that corresponds to connected components of $G_S[V \setminus T]$.

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be the partition of $U$ given by phase one — that is if we denote the connected components of $F$ by $C_1, C_2, \ldots, C_m$, then $P_i = C_i \cap U$ for $1 \leq i \leq m$. Note that the $P_i$s are non–empty, since if some connected component of $F$ contained no vertex from $U$, then it would be removed from $F$ completely during the third step of phase one. Now we proceed as follows:

1. Branch into all possible selections of at most $k - |R|$ edges from $F$;
2. Let $\mathcal{P}'$ denote the partition of $U$ given (as above) by $F$ with the selected edges removed;
3. For each partition $\mathcal{P}''$ of $U$ being at the same time a subpartition of $\mathcal{P}$ and a superpartition of $\mathcal{P}'$, start phase three with partition pair $(\mathcal{P}'', R)$.

We want to check how many times each application of phase two enters phase three. As $F$ has at most $2|U| \leq 2|V(S)|$ edges, we have at most $(2|V(S)|)^k$ choices in the first step. Now, consider a connected component $C_i$ of $F$, from which we removed $s_i$ edges. There are at most $s_i+1$ elements of $\mathcal{P}'$ which are subsets of $C_i$. The number of ways to combine these elements into a partition of $C_i \cap U$ is at most the $(s_i+1)$–st Bell number $B_{s_i+1} \leq (s_i+1)^{s_i+1}$. The product of these is bounded by

$$\prod_i B_{s_i+1} \leq \prod_i (s_i+1)^{s_i+1} = \exp\left(\sum_i (s_i+1)\log(s_i+1)\right)$$
$$\leq \exp\left(\left(1+\sum_i s_i\right)\log\left(1+\sum_i s_i\right)\right) = (1+k)^{1+k},$$

where the last inequality follows from the standard Jensen's inequality corollary $f(\sum_i x_i) \geq \sum_i f(x_i)$ for a convex function $f$ with $f(0) = 0$ and non–negative $x_i$ applied for $f(x) = (1+x)\log(1+x)$. Thus for each execution of phase two, phase three is executed at most $(2|V(S)|)^k(k+1)^{k+1}$ times.

Phase three works as follows:

1. Take the partition pair $(\mathcal{P}'', R)$ (i.e., $R \subseteq V$ and $\mathcal{P}''$ is a partition of $U = V(S) \setminus R$) given by phase two, form a multigraph on the set $\{P_1, P_2, \ldots, P_m\}$ by adding an edge $P_iP_j$ for every edge $uv \in S$, where $u \in P_i$, $v \in P_j$. If any of these edges is not a bridge in the multigraph, return a negative answer from this branch;

2. Otherwise create a graph $G'$ by adding to $G$ a vertex $w_i$ for every $P_i \in \mathcal{P}''$, $i \geq 1$ and adding edges connecting $w_i$ to $u_j$ for each $u_j \in P_i$; let $W = \{w_i\}_{i=1}^m$;

3. Apply Node Multiway Cut to $(G'_S[(V \cup W) \setminus R], W, k - |R|)$. If it returns a negative answer, we return a negative answer from this branch, otherwise we return $Q \cup R$ as a solution, where $Q$ is the set returned by Node Multiway Cut.

The execution of this phase takes $O^\star(2^k)$ time [46]. Thus, as the first phase branches out into at most $(2|V(S)| + 1)^k$ executions of phase two, phase two branches out into at most $(2|V(S)|)^k(k+1)^{k+1}$ instances of phase three, and phase three executes in $O^\star(2^k)$, the running time of the whole algorithm is at most $O^\star(2^{O(k \log |V(S)|)})$ (we may assume $k \leq |V(S)|$ — otherwise, removing $V(S)$ from $V$ gives a trivial positive solution).

Now we prove the correctness of this algorithm. First assume the algorithm returns a solution. It was then found by phase three. That is, for some partition pair $(\mathcal{P}'', R)$ the Node Multiway Cut instance $(G'_S[(V \cup W) \setminus R], W, k - |R|)$ has a solution $Q \subseteq V \setminus R$, $|Q| \leq k - |R|$. We claim that $Q \cup R$ is a solution to the initial Edge-Subset-FVS instance $(G, S, k)$. Clearly $|Q \cup R| = |Q| + |R| \leq k$. We prove there are no simple cycles through edges of $S$ in $G'[(V \cup W) \setminus (Q \cup R)]$, which implies the same for its subgraph $G[V \setminus (Q \cup R)]$.

Let $C_i$ be a connected component of $G'_S[(V \cup W) \setminus (Q \cup R)]$ that contains $w_i$. As $Q$ is a solution to the Node Multiway Cut instance $(G'_S[(V \cup W) \setminus R], W, k - |R|)$, $C_i \neq C_j$ for $i \neq j$ and, by construction of the set $W$, $(P_i \setminus Q) = V(C_i) \cap S$. Let $uv \in S$ be an edge of $G'[(V \cup W) \setminus (Q \cup R)]$, i.e., $u, v \notin Q \cup R$. As $u, v \in V(S) \setminus R$, $u \in P_i$ and $v \in P_j$ for some $1 \leq i, j \leq m$ (recall that $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$). If $i = j$, the edge $uv \in S$ would create a self-loop at vertex $P_i$ in the multigraph considered in phase three and the partition pair $(\mathcal{P}'', R)$ would be rejected. We infer that $i \neq j$, and $uv$ connects $C_i$ and $C_j$.

By contradiction, assume that there is an $S$-cycle $\mathcal{C}$ in $G'[(V \cup W) \setminus (Q \cup R)]$. The cycle $\mathcal{C}$ consists of edges from $S$, that connect different components $C_i$, and edges not from $S$, that have both endpoints in the same connected component of $G'_S[(V \cup W) \setminus (Q \cup R)]$. The cycle $\mathcal{C}$ contains at least one edge from $S$, and the edges of $S$ are exactly these edges of $G'[(V \cup W) \setminus (Q \cup R)]$ that have endpoints in different connected components of $G'_S[(V \cup W) \setminus (Q \cup R)]$. We infer that $\mathcal{C}$ contains vertices only from the connected components $C_i$ for $1 \leq i \leq m$, and visits at least two such components. Thus the edges from $S$ on $\mathcal{C}$ form a cycle in the multigraph considered in phase three, and the partition pair $(\mathcal{P}'', R)$ would be rejected. We infer that any solution found by our algorithm is indeed a solution of the Edge-Subset-FVS problem.

On the other hand, assume there exists some solution $T$ of the Edge-Subset-FVS problem. We show how our algorithm arrives at a positive answer in this case. In the first phase, if any vertex $t \in T$ remains in the forest $F$ constructed in this phase in Point 4, we select the branch that adds $t$ to $R$. If no vertex from $T$ remains in $F$ at some iteration of the first phase, we branch out to the second phase. Note that $R \subseteq T$. In the first step of the second phase, we choose the edges into which the vertices from $T$ were contracted in Point 4 of the first Phase (if some were dropped due to being leaves or isolated vertices in $F$, we simply choose fewer edges). Now consider the partition $\mathcal{P}''$ of $U = V(S) \setminus R$ given by the relation of being in the same connected component of $G_S[V \setminus T]$. It is a subpartition of $\mathcal{P}$, as $\mathcal{P}$ is simply the partition given by $G_S[V \setminus R]$, and $R \subseteq T$. It is also a superpartition of $\mathcal{P}'$, as $\mathcal{P}'$ is given by removing the whole $T$ and all the edges which are not edges of the forest $F$. Thus $\mathcal{P}''$ is one of the partitions considered in the second phase, and thus enters the third phase. Now for this partition the edges of $S$ are bridges in the sense of the first step of phase three, as the vertices of the graph considered there correspond exactly to connected components of $G_S[V \setminus T]$, and $G[V \setminus T]$ has no simple cycles through edges in $S$. Moreover, $T \setminus R$ is a solution for the Node Multiway Cut problem by its definition. Thus Node Multiway Cut returns some positive answer (not necessarily $T \setminus R$) in this branch, and thus the algorithm gives the correct answer. □

### 3.2.4 The relationship of Subset-FVS and terminal separation

It is known (e.g. [63]) that in the weighted case the Node Multiway Cut problem with deletable terminals (i.e., the solution set may include some terminals) can be reduced to weighted Subset-FVS by adding a vertex $s$ (with infinite weight) to the graph and connecting it to all the terminals, where $S = \{s\}$. Here we present a modified version, adjusted to the unweighted parameterized setting. Both the node and edge versions of the Multiway Cut problem are known to be FPT since 2004 [110].

**Theorem 3.21.** *An instance $(G, \mathcal{T}, k)$ of the* Node Multiway Cut *problem with deletable terminals can be transformed in polynomial time into an equivalent instance $(G', S, k)$ of the* Edge Subset Feedback Vertex Set *problem.*

*Proof.* Let $\mathcal{T} = \{v_1, \ldots, v_t\}$. We add a set $\mathcal{T}' = v'_1, \ldots, v'_t$ of $t$ vertices to the graph $G$. Together with the vertices from the set $\mathcal{T}'$ we add a set of edges $S = \{v_i v'_i : 1 \leq i \leq t\}$. Moreover, we add an edge between every pair of vertices from the set $\mathcal{T}'$ so that $G[\mathcal{T}']$ becomes a clique. Denote the resulting graph by $G'$. Assume that $(G, \mathcal{T}, k)$ is a YES-instance of Node Multiway Cut where $T \subseteq V$ is a solution. Clearly $T$ is a solution for the instance $(G', S, k)$ of Edge-Subset-FVS since an $S$-cycle in $G'[(V \cup \mathcal{T}') \setminus T]$ implies a path between terminals in $G[V \setminus T]$ (see Figure 3.6).

In the other direction, assume that $(G', S, k)$ is a YES-instance of Edge-Subset-FVS where $T \subseteq V$ is a set of removed vertices. Let $T' = T \setminus \mathcal{T}' \cup \{v_i : v'_i \in T\}$. We now prove that $T'$ is a solution for the Node Multiway Cut instance $(G, \mathcal{T}, k)$. Clearly $|T'| \leq |T| \leq k$. Assume that there exists a path $P$ in $G[V \setminus T']$ between terminals $v_{i_1}$ and $v_{i_2}$. In particular, this means that $v_{i_1}, v_{i_2} \notin T'$, so $v_{i_1}, v_{i_2}, v'_{i_1}, v'_{i_2} \notin T$. Thus the path $P$ together with the path $v_{i_1} v'_{i_1} v'_{i_2} v_{i_2}$ forms an $S$-cycle that is not hit by $T$, a contradiction. □

## 3.3 Lower bounds in kernelization

Up to 2007, many kernelization algorithms with good guarantees on the output size have been discovered. However, many problems, including Longest Path, eluded efforts of the researchers, and it was conjectured that such algorithm might not exist. As pointed out in the survey of Guo and Niedermeier

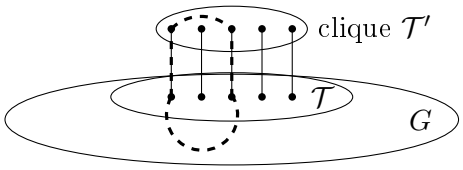


Figure 3.6: Reduction used in Theorem 3.21.

.

from 2007 [88], the field of kernelization was lacking a framework for proving lower bounds, in particular, for excluding the existence of polynomial kernels.

Such a framework was discovered in 2008 by Fortnow and Santhanam [79] and Bodlaender et al. [15]. The key idea is the following definition of a *composition algorithm* (in this section we follow the notation from [55]):

**Definition 3.22** (Composition [15, 55])**.** A *composition algorithm* for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $(x_1, k)$, $(x_2, k)$, ..., $(x_t, k)$ with $(x_i, k) \in \Sigma^* \times \mathbb{N}$ for each $1 \leq i \leq t$, uses polynomial time in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}$ with $(y, k') \in Q$ iff $\exists_{1 \leq i \leq t}(x_i, k) \in Q$ and $k'$ is polynomial in $k$. A parameterized problem is called *compositional* if there is a composition algorithm for it.

Let us now make an example of the Longest Path problem, one of the first problems proven to be hard to polynomially kernelize (the example is based on the work of Bodlaender et al. [15]).

| Longest Path |
| --- |
| **Input:** An undirected graph $G = (V, E)$ and an integer $k$. |
| **Parameter:** $k$. |
| **Question:** Does there exist a simple path of length $k$ in $G$? |

Longest Path has trivial composition algorithm: given instances $(G_1, k)$, $(G_2, k)$, ..., $(G_t, k)$, output an instance $(G, k)$, where $G$ is a disjoint union of the graphs $G_i$. Clearly, $G$ contains a simple path of length $k$ if and only if one of the input graphs contains it, too.

Assume now that Longest Path admits a kernelization algorithm that reduces the size of the instance to a one represented by at most $g(k)$ bits, where $g$ is a polynomial. If $t > g(k)$, then in the output there is less than one bit per an input instance. Thus, in some sense, the kernelization algorithm needs to solve large part of the input instances $(G_i, k)$, which, for

$NP$-complete problem LONGEST PATH, should be impossible. This intuition, given by Bodlaender et al. [15], was proved to be true by Fortnow and Santhanam [79].

Before we state formally the aforementioned result, let us introduce one more notation: given a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$, its unparameterized version is a language $\tilde{Q} = \{x\#1^k : (x,k) \in Q\}$, i.e., we append the parameter written in unary.

**Theorem 3.23** ([15, 79]). *Let $Q$ be a compositional parameterized problem whose unparameterized version $\tilde{Q}$ is NP-complete. Then, unless NP $\subseteq$ coNP/poly, there is no polynomial kernel for $Q$.*

We note that NP $\subseteq$ coNP/poly causes a collapse of the polynomial hierarchy up to its third level [26, 131].

To prove the non-existence of a polynomial kernel for some parameterized problem, it is not necessary to go through Theorem 3.23. As in the case of NP-complete problems, we can use reductions instead (first used in [17]).

**Definition 3.24** ([17, 55]). Let $P$ and $Q$ be parameterized problems. We say that $P$ is polynomial parameter reducible to $Q$, written $P \leq_{Ptp} Q$, if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ and a polynomial $p$, such that for all $(x,k) \in \Sigma^* \times \mathbb{N}$ the following holds: $(x,k) \in P$ iff $(x',k') = f(x,k) \in Q$ and $k' \leq p(k)$. The function $f$ is called a *polynomial parameter transformation*.

**Theorem 3.25** ([17, 55]). *Let $P$ and $Q$ be parameterized problems and $\tilde{P}$ and $\tilde{Q}$ be the unparameterized versions of $P$ and $Q$ respectively. Suppose that $\tilde{P}$ is NP-hard and $\tilde{Q}$ is in NP. Assume there is a polynomial parameter transformation from $P$ to $Q$. Then if $Q$ admits a polynomial kernel, so does $P$.*

*Proof.* Let $\mathcal{A}_Q$ be a kernelization algorithm for the language $Q$ with polynomial (in the parameter) guarantee on the size of the output. Assume we are given an instance $(x_P, k_P)$ of the problem $P$. Consider the following kernelization algorithm.

1. Use the assumed polynomial parameter transformation from $P$ to $Q$, to obtain an instance $(x_Q, k_Q)$ of problem $Q$, equivalent to $(x_P, k_P)$ and with $k_Q$ bounded polynomially in $k_P$.

2. Apply the algorithm $\mathcal{A}_Q$ to the instance $(x_Q, k_Q)$ to obtain an equivalent instance $(x'_Q, k'_Q)$. Note that now both $|x'_Q|$ and $k'_Q$ are bounded polynomially in $k_P$.

3. As $\tilde{P}$ is NP-hard, and $\tilde{Q}$ is in NP, there exists a Karp reduction from $\tilde{Q}$ to $\tilde{P}$. Use it on the unparameterized version of $(x'_Q, k'_Q)$ to obtain an unparameterized version of an instance $(x'_P, k'_P)$, equivalent to $(x'_Q, k'_Q)$. Note that, again, both $|x'_P|$ and $k'_P$ are bounded polynomially in $k_P$, and we have the desired kernelization algorithm for $P$.

□

The result of Fortnow and Santhanam [79] was further improved by Dell and van Melkebeek [51] to allow excluding polynomial kernels of particular exponent: for example, they show that Vertex Cover does not admit a kernel with $O(k^{2-\varepsilon})$ edges for any $\varepsilon > 0$ (unless NP $\subseteq$ coNP/poly). Moreover, the idea of composition was later improved to a more robust *cross-composition* [16]. However, in the following section the standard definition of composition presented above is sufficient for our needs.

It is worth noting that, as pointed out by Kratsch [102], the framework of composition algorithms is a bit stronger than what was required: the existence of a composition algorithm rules out not only deterministic polynomial-time kernelization, but also a co-nondeterministic one. (A co-nondeterministic kernel is a nondeterministic polynomial-time algorithm that outputs a NO-instance on at least one computation path if and only if the input instance is a NO-instance.) However, up to today we do not know any example of an (important and motivated) problem that admits a co-nondeterministic kernel, but eludes attempts to construct a deterministic one.

A composition algorithm (Definition 3.22) is often called an OR-composition in literature, as opposed to an AND-composition, where the output instance is required to be a YES-instance if and only *all* input instances are YES-instances. Basing on the same intuition as described in this section, an NP-complete problem should not admit both a polynomial kernel and an AND-composition. However, it is now a major open problem in the field of kernelization to support this claim with a proof based on some widely-believed complexity assumption.

## 3.4 Lower bounds on kernelization of connectivity problems in degenerate graphs

One of the most seminal discoveries in parameterized complexity in the last decade was that many problems which are hard in general graphs — i.e. without a polynomial kernel or even not FPT — have small kernels in sparse graph classes, such as planar graphs, bounded genus graphs, apex-minor-free graphs or $H$-minor-free graphs. The first result of this type was a linear kernel for Dominating Set in planar graphs, due to Alber et al. [2]. Their idea of a *region decomposition* turned out to be very robust, and applicable not only to different problems, but also to wider graph classes than the planar ones. Recent results include linear kernels for Dominating Set and Connected Dominating Set in apex-minor-free graphs and linear kernels for Feedback Vertex Set and Connected Vertex Cover in $H$-minor-free graphs [78].

One should note that all the aforementioned results are applicable to problems, where the solution is in some sense *dense* in the input graph (e.g., every vertex of a graph is at distance at most one from a dominating set in a graph). Currently it is an open problem to show a polynomial kernel for Directed Feedback Vertex Set (delete $k$ vertices from a directed graph to obtain an acyclic graph), even in planar graphs.

Sparse graph classes are also important from the point of view of fixed-parameter algorithms, not only kernels. The theory of bidimensionality [54] allows to construct algorithms subexponential in $k$ on planar, bounded genus or $H$-minor-free graphs for many important problems, including Dominating Set, Feedback Vertex Set or Independent Set. The core part of this approach is the grid-minor theorem of Robertson and Seymour, which, for sparse graph classes, gives a linear dependency between the treewidth of a graph and the size of the largest grid that the graph contains as a minor. We also note that evaluation of first-order formulas in sparse graphs is fixed-parameter tractable when parameterized by the length of the formula; this holds even for graph classes strictly larger than those excluding a fixed minor [60].

The aforementioned results use the topological structure of the considered graph classes. However, sometimes an even weaker assumption on the graph class leads to significantly better algorithms and kernels than in the general case. One may, for instance, consider the class of $d$-degenerate graphs. A

graph is called $d$-degenerate if its every induced subgraph contains a vertex of degree at most $d$[1]. For instance, the class of 1-degenerate graphs is the class of forests, and all planar graphs are 5-degenerate. Moreover, every $H$-minor-free graph is $d$-degenerate, where the constant $d$ depends on the minor [101, 124, 125]. On the other hand, if we take an arbitrary graph and subdivide every its edge, we obtain a 2-degenerate graph. Thus, bounded degeneracy graphs exhibit the same properties with respect to degrees of vertices as $H$-minor-free graphs, but do not have any topological properties.

Alon and Gutner [4] followed by Golovach and Villanger [83] proved that Dominating Set and Connected Dominating Set, which are $W[2]$-complete (i.e., very unlikely to be FPT) in general graphs [56], become FPT when the input graph is $d$-degenerate. Recently, Philip et al. [116] proved that Dominating Set is FPT and admits a polynomial kernel in a larger class of graphs: graphs excluding the biclique $K_{i,j}$ as a subgraph (note that a $d$-degenerate graph cannot contain $K_{d+1,d+1}$ as a subgraph). This result generalizes all previously known fixed-parameter algorithms for Dominating Set (one of the core problems in parameterized complexity and perhaps the most natural of the $W[2]$-complete problems) in different graph classes.

A natural question arises: since we were so successful with the $H$-minor-free graph classes, does the bounded degeneracy assumption help in the kernelization of more problems? In particular, the question of finding a polynomial kernel for Connected Dominating Set in $d$-degenerated graphs was posed at the 1st Workshop on Kernels (WORKER'09, Bergen, Norway). We note that problems involving connectivity requirement, such as Connected Dominating Set or Connected Vertex Cover, are usually much more difficult to kernelize to than their non-connected counterparts, as many greedy reduction rules are not applicable due to the connectivity requirement. For example, when solving the Vertex Cover problem, we can forget about vertices, whose all incident edges were already covered; in the Connected Vertex Cover problem, where we additionally require that the solution induces a connected subgraph, such a vertex may be used to ensure connectivity.

In this section we provide a few negative answers to questions about the existence of polynomial kernels for connectivity problems in graphs of

[1] In literature, the definition of a $d$-degenerate graph is sometimes a bit different: it is required that all subgraphs have a vertex of degree at most $d$, not only the induced ones. However, it is easy to see that these two definitions are equivalent.

bounded degeneracy. Note that this is in sharp contrast with the existence of the linear kernel for Connected Dominating Set in apex-minor-free graphs [78].

Similarly as when showing that a problem is NP-hard, when proving kernelization lower bounds the main difficulty is to use a convenient intermediate problem. The main contribution in this section is the idea to use the Colourful Graph Motif problem, which, intuitively, encapsulates the hardness of the connectivity requirement.

Colourful Graph Motif
**Input:** A graph $G = (V, E)$, an integer $k$ and a function $f : V \to \{1, 2, \ldots, k\}$.
**Parameter:** $k$.
**Question:** Does there exist a connected set $S \subseteq V$ of cardinality $k$, such that $f|_S$ is bijective?

We think of the function $f$ to be a colouring of $V$ — each number from $\{1, 2, \ldots, k\}$ corresponds to a single colour — and we ask whether it is possible to choose a connected set containing exactly one vertex of each colour.

Fellows et al. [69] have shown that, surprisingly, this problem is NP-hard even in the class of trees of maximum degree 3. We extend this analysis by showing NP-hardness and nonexistence of a polynomial kernel (unless NP $\subseteq$ coNP/poly) for Colourful Graph Motif in comb graphs, a subclass of trees of maximum degree three (see Definition 3.31). A significantly deeper discussion on the hardness of Colourful Graph Motif problem in different classes of trees can be found in a subsequent work by Ambalath et al. [6].

The Colourful Graph Motif problem is simple enough to admit a (polynomial parameter) reduction to Connected Dominating Set in 2-degenerate graphs. As a by-product of this analysis, we obtain an alternative proof that Steiner Tree does not admit a polynomial kernel in arbitrary graphs. The original proof, via reduction from Red Blue Dominating Set (aka Set Cover) is due to Dom et al. [55]. We analyze Colourful Graph Motif in Section 3.4.2 and apply it to Connected Dominating Set to show that Connected Dominating Set does not admit a polynomial kernel in 2-degenerate graphs. In Section 3.4.2 we also show the reduction from Colourful Graph Motif to Steiner Tree.

As a warmup, in Section 3.4.1 by easy reductions and using already known results we show that Steiner Tree, Connected Feedback Vertex Set

and Connected Odd Cycle Transversal do not admit polynomial kernels in 2-degenerate graphs. All discussed problems are parameterized by the solution size, except for Steiner Tree, which is parameterized both by the solution size and the size of the terminal set. Precise definitions of considered problems can be found in appropriate sections. For a graph problem $Q$, by $d$–deg–$Q$ we denote the problem $Q$ with input restricted to $d$-degenerate graphs.

### 3.4.1 Easy cases

We begin by showing that, unless NP $\subseteq$ coNP/poly, no polynomial kernel exists even for 2-degenerate graphs for three problems: Steiner Tree, Connected Feedback Vertex Set and Connected Odd Cycle Transversal. We use the results of Dom et al. [55], where the authors show that Steiner Tree and Connected Vertex Cover do not admit a polynomial kernel in the class of all graphs.

**Theorem 3.26** (Dom et al. [55])**.** *The* Steiner Tree *problem, parameterized by the solution size and the size of the terminal set, and the* Connected Vertex Cover *problems, parameterized by the solution size, do not admit polynomial kernels unless NP $\subseteq$ coNP/poly.*

We use these two problems as starting points for our reductions, making use of Theorem 3.25. The presented constructions are adjustments of reductions made for Connected Feedback Vertex Set in general graphs [112].

First, let us recall precise definitions of considered problems.

Steiner Tree (ST)
**Input:** A graph $G = (V, E)$, a set of terminals $T \subseteq V$ and an integer $k$.
**Parameter:** $t := |T|$ and $k$.
**Question:** Does there exist $S \subseteq V$, such that $G[S \cup T]$ is connected and $|S| \leq k$?

Connected Feedback Vertex Set (CFVS)
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Parameter:** $k$.
**Question:** Does there exist a set $S \subseteq V$ of cardinality at most $k$, such that $G[S]$ is connected and $G[V \setminus S]$ contains no cycles?

| Connected Odd Cycle Transversal |
|---|
| **Input:** A graph $G = (V, E)$ and an integer $k$. **Parameter:** $k$. **Question:** Does there exist a set $S \subseteq V$ of cardinality at most $k$, such that $G[S]$ is connected and $G[V \setminus S]$ is bipartite (that is, contains no cycles of odd length)? |

| Connected Vertex Cover |
|---|
| **Input:** A graph $G = (V, E)$ and an integer $k$. **Parameter:** $k$. **Question:** Does there exist a set $S \subseteq V$ of cardinality at most $k$, such that $G[S]$ is connected and every edge $e \in E$ has at least one endpoint in $S$? |

Now let us note the following simple observation.

**Lemma 3.27.** *Assume that in a graph $G$ every edge has an endpoint of degree at most 2. Then $G$ is 2-degenerate.*

*Proof.* Consider an induced subgraph of $G$. If it does not contain any edge, there is nothing to prove. Otherwise, it has at least one edge, which by assumption admits at least one endpoint of degree at most two in $G$. The claim follows from the fact that the degree of a vertex does not increase in taking induced subgraphs. □

We now show reductions to each of the three aforementioned problems.

**Proposition 3.28.** *There exists a polynomial parameter transformation from* Connected Vertex Cover *(in general graphs) to* 2*–deg–*Connected Feedback Vertex Set*, as well as from* Connected Vertex Cover *to* 2*–deg–*Connected Odd Cycle Transversal*.*

*Proof.* Consider any instance $(G, k)$ of Connected Vertex Cover. We create a graph $G' = (V', E')$. We take $V' = V \cup E_1 \cup E_2 \cup E_3$ — the vertices of $G'$ are the vertices of $G$ plus three new vertices $e_1, e_2, e_3$ for each edge $e$ of $G$. For each edge $e = uv \in E$ we add five edges to $E'$: $ue_1, e_1v, ve_2, e_2e_3$ and $e_3u$. This means we transform each edge of $G$ into a cycle of length 5, where the original vertices are not adjacent on the cycle. Lemma 3.27 implies that $G'$ is 2-degenerate. We claim that the Connected Vertex Cover instance $(G, k)$, a 2–deg–Connected Feedback Vertex Set instance $(G', 2k-1)$ and a 2–deg–Connected Odd Cycle Transversal instance $(G', 2k-1)$

are equivalent. We prove this by showing three implications which form a cycle.

First, let us assume that the answer to the Connected Vertex Cover instance $(G, k)$ is positive and let $S$ be a connected vertex cover of $G$ of size at most $k$. As $S$ is connected, we can create a spanning tree $T$ in $G[S]$, that consists of at most $k-1$ edges $E_S \subseteq E$. Let $E'_S = \{e_1 : e \in E_S\}$ — that is, for each edge $e \in E_S$ we take into $E'_S$ the vertex in $V'$ corresponding to $e$ that is adjacent in $G'$ to both endpoints of $e$ in $G$. Let $S' = S \cup E'_S$. Note that $G'[S']$ is isomorphic to a graph obtained from $T$ by subdividing every edge once, thus it is connected. We claim that $G'[V' \setminus S']$ contains no cycles. Assume that $C$ is a cycle in $G'[V' \setminus S']$. $C$ cannot consist only of elements of $V$ (since $V$ is independent in $G'$), thus $C$ contains some element $e_i$. As for each $e \in E$ the vertices $e_1$, $e_2$ and $e_3$ are of degree two, and $e_2e_3 \in E'$, $C$ also has to contain both vertices from $V$ which the corresponding edge $e \in E$ connects. This, however, means in particular that neither of these vertices was in $S$, which is a contradiction with the assumption that $S$ was a vertex cover of $G$, as the edge $e$ is not covered. We infer that the answer to the Connected Feedback Vertex Set instance $(G', 2k-1)$ is positive.

Clearly, any connected feedback vertex set is also a connected odd cycle transversal. Thus, if the answer to the 2–deg–Connected Feedback Vertex Set instance $(G', 2k-1)$ is positive, so is the answer to the 2–deg–Connected Odd Cycle Transversal instance $(G', 2k-1)$.

Now assume we have a minimum connected odd cycle transversal $S \subseteq V'$ in $G'$ of cardinality at most $2k-1$. Assume $|S| \geq 2$ (the case $|S| = 1$ is trivial). Notice that $|S \cap V| \leq k$ — if we have more than $k$ vertices from $V$, they form at least $k+1$ connected components, and each vertex from $E_1 \cup E_2 \cup E_3$ connects at most two of them — thus $S$ would not be connected. We claim $S \cap V$ forms a connected vertex cover of $G$. First note that $G[S \cap V]$ is connected: if $v, v' \in S \cap V$ are connected in $G'[S]$ by a path $P$, the vertices from $V$ on $P$ form a path connecting $v$ and $v'$ in $G[S \cap V]$. Furthermore, consider any edge $e = uv$ in $E$ and the corresponding cycle $(u, e_1, v, e_2, e_3)$ in $G'$. As $S$ is an odd cycle transversal in $G'$, at least one of these five vertices must belong to $S$. As $|S| \geq 2$ and $S$ is connected, unless $S = \{e_2, e_3\}$, at least one of $u, v$ is in $S$ — and thus $e$ is covered in $G$ by $S \cap V$. In the case $S = \{e_2, e_3\}$, note that $\{e_2\}$ is also a connected odd cycle transversal of $G'$, contradicting the minimality of $S$. □

**Corollary 3.29.** *For all $d \geq 2$, the problems $d$–deg–*Connected Feed-

BACK VERTEX SET *and* $d$*–deg–*CONNECTED ODD CYCLE TRANSVERSAL *do not admit a polynomial kernel unless* $NP \subseteq coNP/poly$.

*Proof.* As the unparameterized version of the CONNECTED VERTEX COVER problem in general graphs is NP-complete [82], by Proposition 3.28 and Theorem 3.25, a polynomial kernel for $d$–deg–CONNECTED FEEDBACK VERTEX SET or $d$–deg–CONNECTED ODD CYCLE TRANSVERSAL would imply a polynomial kernel for CONNECTED VERTEX COVER in general graphs. By the result of Dom et al. [55] (Theorem 3.26), this would imply NP $\subseteq$ coNP/poly and a collapse of the polynomial hierarchy up to its third level. □

The last reduction to degenerate graphs from previously known results is for 2-deg-STEINER TREE. The alternative proof of the kernelization hardness of 2-deg-STEINER TREE, via reduction from COLOURFUL GRAPH MOTIF, can be found in Section 3.4.2.

**Proposition 3.30.** *These is a polynomial parameter transformation from* STEINER TREE *to* 2*-deg-*STEINER TREE*, and* $d$*-deg-*STEINER TREE *for all* $d \geq 2$ *admits no polynomial kernel unless* $NP \subseteq coNP/poly$.

*Proof.* Take a general instance $(G, k, T)$ of STEINER TREE. Create a new graph $G'$ by subdividing each edge — formally, let $V' = V \cup E$ and $ve \in E'$ if $v$ is an endpoint of $e$ in $G$. The graph $G'$ is 2–degenerate by Lemma 3.27.

We claim that the answer for $(G, k, T)$ is the same as the answer for $(G', 2k+|T|-1, T)$. Assume we have a solution $S$ of $(G, k, T)$. Then $G[S \cup T]$ is connected. Take any spanning tree of $G[S \cup T]$ and let $F$ be the set of its edges. We have $|F| \leq k + |T| - 1$. Note that $F \cup S$ is a solution in $(G', 2k + |T| - 1, T)$. In the other direction, if we have a solution $S'$ in $(G', 2k+|T|-1, T)$, we consider $S = S' \cap V$. Note that $S \cup T$ has cardinality at most $k + |T|$ — since $|S' \cup T| \leq 2k + 2|T| - 1$, $S \cup T$ is isolated in $G'$, and adding a single vertex from $E$ connects at most two components of the set. Thus $S$ has a cardinality at most $k$, and $G[S \cup T]$ is connected (for otherwise $S' \cup T$ could not be connected in $G'$).

Hence we proved that there is a polynomial parameter transformation from STEINER TREE to 2–deg–STEINER TREE. The second part of the claim follows similarly as in the proof of Corollary 3.29. □

### 3.4.2 From Colourful Graph Motif to Connected Dominating Set

The Connected Vertex Cover problem is, in a number of cases, too specific to allow easy reductions. The Colourful Graph Motif problem turned out to be very handy in our case.

Fellows et al. [69] have shown that Colourful Graph Motif in the class of trees of maximum degree 3 is already NP-complete. We extend this analysis by showing that Colourful Graph Motif is NP-complete in comb graphs.

**Definition 3.31.** A graph $G = (V, E)$ is called a *comb graph* if it is a tree, all vertices are of degree at most 3, and all the vertices of degree 3 lie on a single simple path. Any such path is called a *spine* of a comb graph.

See Figure 3.7 for an example of a comb graph.

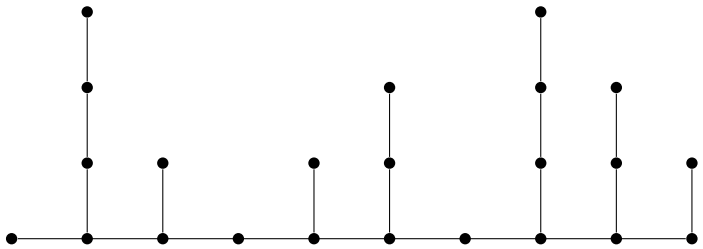

Figure 3.7: An example of a comb graph.

**Proposition 3.32.** *The unparameterized version of the* Colourful Graph Motif *problem in comb graphs is NP–complete.*

*Proof.* Being in NP is obvious. For hardness, we reduce the CNF–SAT problem to Colourful Graph Motif in a comb graph. Let us consider an instance $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ of CNF–SAT with variables $x_1, x_2, \ldots, x_n$. We create a comb graph $G = (V, E)$. For each clause $C_r = (l_1^r \vee l_2^r \vee \ldots \vee l_{k_r}^r)$ we add $4k_r$ vertices to $G$: $v(C_r, l_i^r, j)$ for $1 \leq i \leq k_r$ and $j = 1, 2, 3, 4$. Also, for each variable $x$ we add four vertices: $v(x, j)$ and $v(\neg x, j)$ for $j = 0, 1$. Finally, we add two vertices $s$ and $t$. The edges are as follows (see Figure 3.8 for an illustration):

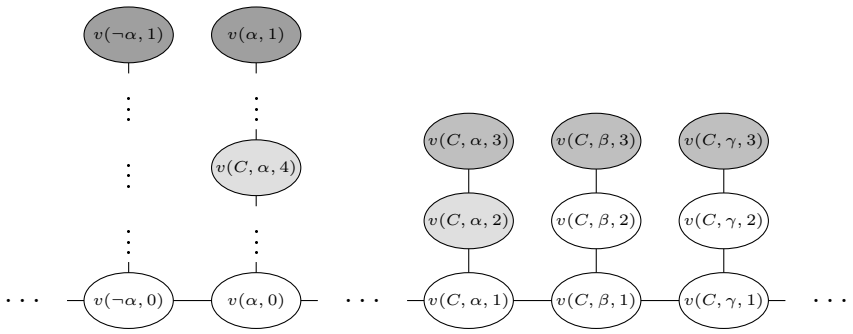


Figure 3.8: Part of the constructed comb illustrating vertices added for clause $C = (\alpha \vee \beta \vee \gamma)$ and their interaction with paths arranged for literals $\alpha$ and $\neg\alpha$.

- For any $C_r$ and any $l_i^r$ in $C_r$ we arrange the vertices $v(C_r, l_i^r, 1)$, $v(C_r, l_i^r, 2)$ and $v(C_r, l_i^r, 3)$ into a path, in this order. By $\mathcal{P}_{\text{clause}}$ we denote the set of all such paths constructed for all clauses $C_r$ and literals $l_i^r$.

- For any variable $x$ of the CNF-formula, for each of the two literals $l = x$ or $l = \neg x$, we arrange the two vertices $v(l, j)$ and all the vertices of the form $v(C_r, l_i^r, 4)$ where $l_i^r = l$ into a single path, such that $v(l, 0)$ and $v(l, 1)$ lie on the ends of this path. By $\mathcal{P}_{\text{literal}}$ we denote the set of all such paths constructed for all literals $l$.

- We arrange the vertices $s$, $t$, $v(l, 0)$ for all literals $l$ and $v(C_r, l_i^r, 1)$ for all clauses $C_r$ and literals $l_i^r$ in $C_r$ into a single path, where $s$ and $t$ are the ends of this path. This path, denoted by $P_{\text{spine}}$, is a spine of the comb.

It is easy to see that the resulting graph is a comb.

Now we create a colouring function $f$. We set $f(v(C_r, l_i^r, 2)) = f(v(C_r, l_i^r, 4))$ for any literal $l_i^r$ in any clause $C_r$, for each clause $C_r$ we set all $f(v(C_r, l_i^r, 3))$ equal for $i = 1, \ldots, k_r$ and we set $f(v(x, 1)) = f(v(\neg x, 1))$ for any variable $x$. All function values of $f$ not stated to be equal are different.

Now let us see how a satisfying assignment of the CNF-formula variables corresponds to a connected set $S$ in $G$. Assume we have a satisfying assignment $\phi$ for the formula $C_1 \wedge C_2 \wedge \ldots \wedge C_m$. We choose $S$ as follows:

- We take all the vertices on the spine of $G$ to be in $S$.

- For each variable $x$, if $\phi(x)$ is true, we choose $v(\neg x, 1) \in S$, and otherwise we choose $v(x, 1) \in S$. In each case, we also take the whole path from $v(\neg x, 0)$ to $v(\neg x, 1)$ or respectively from $v(x, 0)$ to $v(x, 1)$ to be in $S$.

- If the literal $l_i^r$ occurs in the clause $C_r$ and was assigned to be true, we choose the vertex $v(C_r, l_i^r, 2)$ to be in $S$.

- Since the assignment satisfied all the clauses then for each clause $C_r$ at least one of the vertices $v(C_r, l_i^r, 2)$ was chosen to be in $S$. Therefore, we choose one of the corresponding vertices $v(C_r, l_i^r, 3)$ to be in $S$.

A direct check shows that the set thus chosen is connected and contains exactly one vertex of each colour — thus $S$ is a solution to the Colourful Graph Motif problem on graph $G$. The correspondence holds also in the reverse direction — if $S$ is a solution of Colourful Graph Motif on $G$ then we construct a satisfying assignment $\phi$ of the formula as follows. Observe that:

- $s, t \in S$ (as they are the only vertices of their respective colours), thus the whole spine of $G$ is a subset of $S$ by the connectivity of $S$;

- for each variable $x$ exactly one of $v(x, 1)$ or $v(\neg x, 1)$ is in $S$. If it is $v(x, 1)$, we assign $\phi(x)$ to be false, otherwise we assign $\phi(x)$ to be true.

- by the connectivity of $S$ all the vertices $v(C_r, l_i^r, 4)$ are in $S$ if $l_i^r$ is assigned to be false;

- for any clause $C_r$ one of the vertices $v(C_r, l_i^r, 3)$ is in $S$, and thus the appropriate $v(C_r, l_i^r, 2)$ is in $S$;

- as all vertices of $S$ are of a different colour, if $v(C_r, l_i^r, 2) \in S$, then $v(C_r, l_i^r, 4) \notin S$, which means $l_i^r$ was assigned to be true, and thus $C_r$ is satisfied by our assignment.

As CNF–SAT is NP–hard and the unparameterized Colourful Graph Motif problem is created in polynomial (even linear) time with respect to the size of the CNF–SAT problem we started with, the unparameterized Colourful Graph Motif problem in combs is NP–complete. □

We now prove the hardness of kernelization of Colourful Graph Motif in comb graphs. First, note that a simple composition algorithm yields the following result.

**Corollary 3.33.** *The problem* Colourful Graph Motif, *when the input graph is restricted to graphs being a disjoint union of combs, does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

*Proof.* We use Theorem 3.23 and take the disjoint union of graphs and the union of the respective colouring functions as the composition algorithm. Note that any feasible solution in the resulting graph is required to induce a connected subgraph and therefore it needs to be contained in one connected component of the input graphs. In the other direction, clearly a solution to one of the input instances induces a solution of the output instance. □

We finish the analysis with a reduction from a disjoint union of combs to a single comb. A similar result was independently obtained by Ambalath et al. [6].

**Proposition 3.34.** *There exists a polynomial parameter transformation from* Colourful Graph Motif *in a disjoint union of combs to* Colourful Graph Motif *in combs, and* Colourful Graph Motif *in the class of comb graphs does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

*Proof.* Assume we have an instance of Colourful Graph Motif in a disjoint union of combs $G$, with $k$ colours. Assume $k \geq 3$ (if $k \leq 2$, Colourful Graph Motif can be trivially solved in polynomial time and hence we create a trivial output instance). We create a single comb with $k$ colours. Let $G_1, G_2, \ldots, G_l$ be the connected components (combs) of $G$. Let $u_i$ and $v_i$ be the endpoints of the spine of $G_i$. We create the graph $G'$ as follows:

- All $G_i$s are subgraphs of $G'$
- For each $i$ we add vertices $u'_i$ and $v'_i$ to $G'$
- We add edges $u_iu'_i$, $v_iv'_i$ for each $i = 1, 2, \ldots, l$ and an edge $v'_iu'_{i+1}$ for each $i = 1, 2, \ldots, l-1$
- $f(u'_i) = f(u_i)$, $f(v'_i) = f(v_i)$.

The graph $G'$ is a comb graph with the spine consisting of all the spines of $G_i$s and all the added vertices. Any solution of Colourful Graph Motif in $(G, k, f)$ has to be contained in a single $G_i$, as it is connected, and thus can be used without changes in $G'$. On the other hand no solution of Colourful Graph Motif in $G'$ contains $u'_i$ or $v'_i$ — since if it contained one of them, say $v'_i$, then it could not contain $v_i$ (as $f(v_i) = f(v'_i)$), it cannot contain $u_{i+1}$ (since it would have to pass through $u'_{i+1}$, which is of the same colour) — thus, due to connectivity, it would contain at most two vertices, while we assumed $k \geq 3$. This implies that a solution to Colourful Graph Motif in $G'$ is contained in one of the $G_i$s, and thus can be used as a solution in $G$. □

**Corollary 3.35.** 1–*deg*–Colourful Graph Motif *does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

*Proof.* We use Theorem 3.25. The unparameterized version of the problem Colourful Graph Motif with the input graphs restricted to combs is NP-complete, and the unparameterized version of the problem Colourful Graph Motif with the input graphs restricted to forests (i.e., 1-degenerate graphs) is in NP. As any instance of Colourful Graph Motif with the input graphs restricted to combs can be treated as an instance of 1–*deg*–Colourful Graph Motif, this gives us trivially that Colourful Graph Motif in combs $\leq_{Ptp}$ 1–*deg*–Colourful Graph Motif and the proof is finished. □

### Reductions

We propose Colourful Graph Motif as a simple tool to prove that various other problems do not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Firstly, to give some intuition on Colourful Graph Motif, let us note that Colourful Graph Motif is a special case of Group Steiner Tree parameterized by the number of groups.

| Group Steiner Tree |
| --- |
| **Input:** A graph $G = (V, E)$, sets of vertices $T_1, \ldots, T_k \subseteq V$ and an integer $p$. |
| **Parameter:** $k$. |
| **Question:** Does there exist $S \subseteq V$, such that $G[S]$ is connected, $\lvert S\rvert = p$ and $S \cap T_i \neq \emptyset$ for $i = 1, \ldots, k$? |

**Proposition 3.36.** *There exists a polynomial parameter transformation from* $d$*–deg–*Colourful Graph Motif *to* $d$*–deg–*Group Steiner Tree.

*Proof.* Assume we have an instance $(G, k, f)$ of $d$–deg–Colourful Graph Motif. We create an instance of $d$–deg–Group Steiner Tree as follows: we keep the graph $G$, we let $p = k$ and take $T_i = f^{-1}(i)$. Now the problem Group Steiner Tree asks whether there exists a connected set $S$ of cardinality $p = k$ which has a non–empty intersection with each $T_i$. As $p = k$, we infer that the intersection with each $T_i$ is to contain exactly one element. This is exactly the question in Colourful Graph Motif, thus the answer to Colourful Graph Motif in $(G, f, k)$ is the same as the answer to Group Steiner Tree in $(G, \{T_i\}_{i=1}^{k}, p, k)$. □

**Corollary 3.37.** Colourful Graph Motif *can be solved in* $2^k n^{O(1)}$ *time and polynomial space.*

*Proof.* We reduce Colourful Graph Motif to Group Steiner Tree as in the proof of Proposition 3.36 and use $2^k n^{O(1)}$-time algorithm described in [112]. □

Our original motivation for analyzing Colourful Graph Motif was the Connected Dominating Set problem.

| Connected Dominating Set |
| --- |
| **Input:** A graph $G = (V, E)$ and an integer $k$ |
| **Parameter:** $k$. |
| **Question:** Does there exist a set $S \subseteq V$ of cardinality at most $k$, such that $G[S]$ is connected and $S$ is a dominating set of $G$? |

**Proposition 3.38.** *For all* $d \geq 2$*, there exists a polynomial parameter transformation from* $(d-1)$*–deg–*Colourful Graph Motif *to* $d$*–deg–*Connected Dominating Set*, and* $d$*–deg–*Connected Dominating Set *admits no polynomial kernel unless* $NP \subseteq coNP/poly$.

*Proof.* We begin with an instance $(G, k, f)$ of $(d-1)$–deg–Colourful Graph Motif. By Corollary 3.37, we may assume $k \geq 2$, otherwise we can solve the input instance in polynomial time. We create a graph $G' = (V', E')$ as follows (see Figure 3.9 for an illustration):

- We begin with $V' = V$ and $E' = E$;

- for each colour $l \in \{1, 2, \ldots, k\}$ we add two vertices $v_l$ and $v'_l$ to $V'$;
- for each colour $l \in \{1, 2, \ldots, k\}$ we add an edge $v_l v'_l$ to $E'$;
- for each vertex $v \in V$ we add an edge $v v_{f(v)}$ to $E'$.

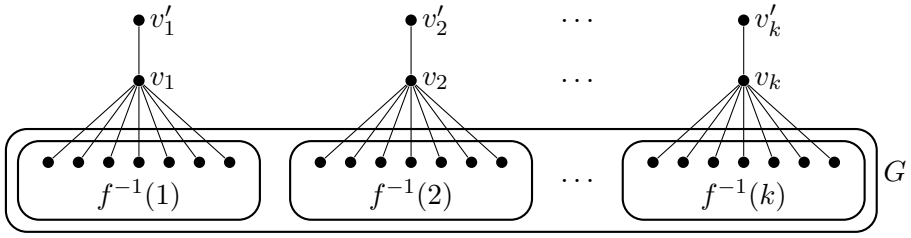


Figure 3.9: Illustration of the reduction from $(d-1)$–deg–Colourful Graph Motif to $d$–deg–Connected Dominating Set (Proposition 3.38).

Firstly, we prove $G'$ is $d$–degenerate. Consider any $S \subseteq V'$. If $S \subseteq V' \setminus V$ then every vertex in $G'[S]$ is of degree at most 1. Otherwise $S \cap V$ is non–empty and $G[S \cap V]$ contains a vertex $v$, which had degree at most $(d-1)$ in $G$, so it has degree at most $d$ in $G'$ (as we added one edge to each vertex of $V$).

Now we prove that the answer to Colourful Graph Motif for $(G, k, f)$ is the same as the answer to Connected Dominating Set for $(G', 2k)$. Assume $k > 1$. If we have a solution $S$ of Colourful Graph Motif in $G$, we create a solution of Connected Dominating Set by putting $S' = S \cup \{v_1, v_2, \ldots, v_k\}$. $S'$ is a dominating set in $G'$, because each vertex $v'_l$ is a neighbour of $v_l$ and each vertex $v \in V$ is a neighbour of $v_{f(v)}$. $G'[S']$ is connected because $G[S]$ is connected and each $v_l$ is adjacent to the vertex of colour $l$ in $S$. On the other hand, any solution $S'$ to Connected Dominating Set in $G'$ has to contain all the vertices $v_l$ (there are two ways to dominate $v'_l$ — either we take $v_l$, or we take $v'_l$, but in the second case we have to take $v_l$ anyway for connectivity). To ensure connectivity we have to take at least one neighbour $u_l$ of each $v_l$ ($u_l \neq v'_l$). As the sets of neighbours of $v_l$s are disjoint and $|S'| \leq 2k$, we infer that exactly one neighbour of each $v_l$ is in $S'$, i.e., $S'$ contains exactly one vertex of each colour. In $G'[S']$ the vertices $v_l$ are of degree 1 (they are not adjacent to each other, and are not

adjacent to $u_j$ for $j \neq l$), thus $G'[S' \setminus \{v_1, v_2, \ldots, v_k\}]$ is connected as $G'[S']$ is connected. We infer that $S' \setminus \{v_1, v_2, \ldots, v_k\}$ is a solution to Colourful Graph Motif in $G$. □

As a final example of the technique we show how to prove that the Steiner Tree problem admits no polynomial kernel in 2-degenerate graphs. The problem was studied in [55], where Steiner Tree was shown to admit no polynomial kernel in general graphs, and a simple reduction to 2-degenerate graphs was shown in Section 3.4.1. We now show a self–contained proof to demonstrate again the applicability of Colourful Graph Motif.

**Proposition 3.39.** *For all $d \geq 2$, here exists a polynomial parameter transformation from $(d-1)$–deg–*Colourful Graph Motif *to d–deg–*Steiner Tree*, and d–deg–*Steiner Tree *admits no polynomial kernel unless NP $\subseteq$ coNP/poly.*

*Proof.* Assume we have an instance $(G, k, f)$ of $(d-1)$-deg-Colourful Graph Motif. We create an instance $(G', T, k)$ of $d$-deg-Steiner Tree as follows: we keep the graph $G$ as the set of non–terminals. Additionally for each colour $i \in \{1, 2, \ldots, k\}$ we add a vertex $t_i$ and edges $vt_i$ for all $v \in f^{-1}(i)$. We ask for a Steiner tree of cardinality $k$ in $G'$ connecting all vertices from $T = \{t_1, t_2, \ldots, t_k\}$ (see Figure 3.10 for an illustration).

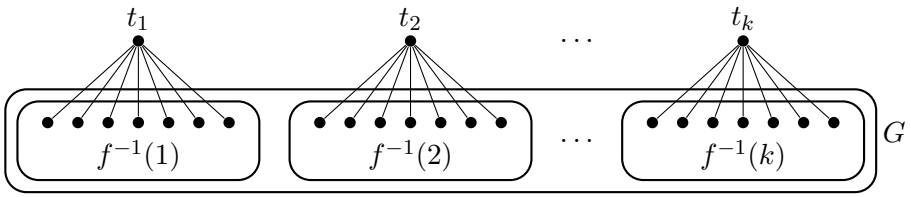


Figure 3.10: Illustration of the reduction from $(d-1)$–deg–Colourful Graph Motif to $d$–deg–Steiner Tree (Proposition 3.39).

First note $G'$ is $d$-degenerate. As in the previous proof, the terminals $T$ form an independent set, while to each non–terminal from $G$ we added exactly one edge. Let $S$ be the solution to Steiner Tree in $G'$. Note that $S$ has to contain exactly one vertex of each colour — if some colour was excluded, the corresponding terminal could not be connected, and the number of colours is at least $|S|$. Moreover, $S$ has to be connected in $G'[V] = G$, as there is only

one vertex of each colour, each terminal is a leaf in the solution, so removing a terminal does not change the connectivity of the solution. On the other hand, it can be also easily seen that any solution of Colourful Graph Motif in $G$ is a solution of Steiner Tree in $G'$. □

# Bibliography


[1] *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010.

[2] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[3] R. B. Allan and R. Laskar. On domination and independent domination numbers of a graph. *Discrete Mathematics*, 23(2):73–76, 1978.

[4] Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

[5] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[6] Abhimanyu M. Ambalath, Radheshyam Balasundaram, Chintan Rao H., Venkata Koppula, Neeldhara Misra, Geevarghese Philip, and M. S. Ramanujan. On the kernelization complexity of colorful motifs. In Venkatesh Raman and Saket Saurabh, editors, *IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 14–25. Springer, 2010.

[7] A. M. Barcalkin and L. F. German. The external stability number of the cartesian product of graphs. *Bul. Akad. Stiinte RSS Moldoven.*, 1:5–8, 1979.

[8] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000.

[9] Daniel Binkele-Raible, Ljiljana Brankovic, Marek Cygan, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Marcin Pilipczuk, Peter Rossmanith, and Jakub Onufry Wojtaszczyk. Breaking the $2^n$-barrier for irredundance: Two lines of attack. *J. Discrete Algorithms*, 9(3):214–230, 2011.

[10] Andreas Björklund. Determinant sums for undirected hamiltonicity. In *FOCS* [1], pages 173–182.

[11] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 67–74. ACM, 2007.

[12] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010.

[13] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[14] Hans L. Bodlaender. On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994.

[15] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[16] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In Thomas Schwentick and Christoph Dürr, editors, *STACS*, volume 9 of *LIPIcs*, pages 165–176. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[17] Hans L. Bodlaender, S. Thomasse, and A. Yeo. Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels, 2008. Technical Report UU-CS-2008-030, Institute of Information and Computing Sciences, Utrecht University, Netherlands.

[18] Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

[19] B. Bollobás and E. J. Cockayne. Graph-theoretic parameters concerning domination, independence, and irredundance. *Journal of Graph Theory*, 3:241–250, 1979.

[20] B. Bollobás and E. J. Cockayne. On the irredundance number and maximum degree of a graph. *Discrete Mathematics*, 49:197–199, 1984.

[21] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In Fortnow and Vadhan [80], pages 459–468.

[22] Boštjan Brešar, Paul Dorbec, Wayne Goddard, Bert L. Hartnell, Michael A. Henning, Sandi Klavžar, and Douglas F. Rall. Vizing's conjecture: a survey and recent results. *Journal of Graph Theory*, pages n/a–n/a, 2011.

[23] Peter Brucker. *Scheduling Algorithms*. Springer, Heidelberg, 2 edition, 1998.

[24] Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly($k$) kernel. In Hans L. Bodlaender and Michael A. Langston, editors, *IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2006.

[25] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

[26] J. Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005.

[27] Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In Kaplan [99], pages 93–104.

[28] M.-S. Chang, P. Nagavamsi, and C. Pandu Rangan. Weighted irredundance of interval graphs. *Information Processing Letters*, 66:65–70, 1998.

[29] Moses Charikar, editor. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. SIAM, 2010.

[30] Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[31] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

[32] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[33] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

[34] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

[35] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In Cynthia Dwork, editor, *STOC*, pages 177–186. ACM, 2008.

[36] Rajesh Chitnis, Mohammadtaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, page to appear, 2012.

[37] W. E. Clark and S. Suen. An inequality related to vizing's conjecture. *Electron. J. Combin.*, 7(Note 4, 3):(electronic), 2000.

[38] E. J. Cockayne, P. J. P. Grobler, S. T. Hedetniemi, and A. A. McRae. What makes an irredundant set maximal? *Journal of Combinatorial Mathematics and Combinatorial Computing*, 25:213–224, 1997.

[39] E. J. Cockayne, S. T. Hedetniemi, and D. J. Miller. Properties of hereditary hypergraphs and middle graphs. *Canadian Mathematical Bulletin*, 21(4):461–468, 1978.

[40] E. J. Cockayne and C. M. Mynhardt. Irredundance and maximum degree in graphs. *Combinatorics, Probability and Computing*, 6:153–157, 1997.

[41] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *Manuscript*, 2011.

[42] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS (to appear)*, 2011. Available at http://arxiv.org/abs/1103.0534.

[43] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Kernelization hardness of connectivity problems in $d$-degenerate graphs. In Dimitrios M. Thilikos, editor, *WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 147–158, 2010.

[44] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than $2^n$. In Camil Demetrescu and Magnús M. Halldórsson, editors, *ESA*, volume 6942 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2011.

[45] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 449–461. Springer, 2011.

[46] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. In *IPEC (to appear)*, 2011. Available at http://arxiv.org/abs/1107.1585.

[47] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Capacitated domination faster than $O(2^n)$. In Kaplan [99], pages 74–80.

[48] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Irredundant set faster than $O(2^n)$. In Tiziana Calamoneri and Josep Díaz,

editors, *CIAC*, volume 6078 of *Lecture Notes in Computer Science*, pages 288–298. Springer, 2010.

[49] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Capacitated domination faster than $O(2^n)$. volume 111, pages 1099–1103, 2011.

[50] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007.

[51] Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Leonard J. Schulman, editor, *STOC*, pages 251–260. ACM, 2010.

[52] Eric D. Demaine, Mohammad Taghi Hajiaghayi, and Dániel Marx. Open problems from dagstuhl seminar 09511, 2009.

[53] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[54] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and $h$-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[55] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009.

[56] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[57] Rod Downey. Parameterized complexity for the skeptic. In *In Proc. 18th IEEE Annual Conference on Computational Complexity*, pages 147–169, 2003.

[58] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

[59] Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors. *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*. Springer, 2004.

[60] Zdenek Dvorak, Daniel Král, and Robin Thomas. Deciding first-order properties for sparse graphs. In *FOCS* [1], pages 133–142.

[61] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[62] Guy Even, Joseph Naor, Baruch Schieber, and Leonid Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. *SIAM J. Discrete Math*, 13(2):255–267, 2000.

[63] Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. *SIAM J. Comput.*, 30(4):1231–1252, 2000.

[64] O. Favaron. Two relations between the parameters of independence and irredundance. *Discrete Mathematics*, 70(1):17–20, 1988.

[65] O. Favaron. A note on the irredundance number after vertex deletion. *Discrete Mathematics*, 121(1-3):51–54, 1993.

[66] O. Favaron, T. W. Haynes, S. T. Hedetniemi, M. A. Henning, and D. J. Knisley. Total irredundance in graphs. *Discrete Mathematics*, 256(1-2):115–127, 2002.

[67] Uriel Feige. Coping with the NP-hardness of the graph bandwidth problem. In Magnús M. Halldórsson, editor, *SWAT*, volume 1851 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2000.

[68] M. R. Fellows, G. Fricke, S. T. Hedetniemi, and D. P. Jacobs. The private neighbor cube. *SIAM Journal on Discrete Mathematics*, 7(1):41–47, 1994.

[69] Michael R. Fellows, Guillaume Fertin, Danny Hermelin, and Stéphane Vialette. Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2007.

[70] Amos Fiat and Peter Sanders, editors. *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*. Springer, 2009.

[71] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1 edition, March 2006.

[72] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: Domination - a case study. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 191–203. Springer, 2005.

[73] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009.

[74] Fedor V. Fomin, Kazuo Iwama, and Dieter Kratsch. Moderately exponential time algorithms, Dagstuhl seminar, 2008.

[75] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1 edition, 2010.

[76] Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In Juraj Hromkovic, Manfred Nagl, and Bernhard Westfechtel, editors, *WG*, volume 3353 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 2004.

[77] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In Charikar [29], pages 748–759.

[78] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In Charikar [29], pages 503–510.

[79] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

[80] Lance Fortnow and Salil P. Vadhan, editors. *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*. ACM, 2011.

[81] Tibor Gallai. Maximum–minimum sätze und verallgemeinerte Faktorem von Graphen. *Acta. Math. Acad. Sci. Hungaricae*, 2:131–173, 1961.

[82] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[83] Petr A. Golovach and Yngve Villanger. Parameterized complexity for domination problems on degenerate graphs. In Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniël Paulusma, editors, *WG*, volume 5344 of *Lecture Notes in Computer Science*, pages 195–205, 2008.

[84] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[85] Fabrizio Grandoni. A note on the complexity of minimum dominating set. *J. Discrete Algorithms*, 4(2):209–214, 2006.

[86] Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011.

[87] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

[88] Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

[89] Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[90] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of domination in graphs*. Marcel Dekker Inc., 1998.

[91] S. T. Hedetniemi, R. Laskar, and J. Pfaff. Irredundance in graphs: a survey. *Congressus Numerantium*, 48:183–193, 1985.

[92] Stephen T. Hedetniemi and Renu C. Laskar. Bibliography on domination in graphs and some basic definitions of domination parameters. *Discrete Mathematics*, 86(1-3):257–277, 1990.

[93] N. Hefetz and I. Adiri. An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, 7:354–360, 1982.

[94] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.

[95] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[96] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In Fortnow and Vadhan [80], pages 313–322.

[97] Yoichi Iwata. A faster algorithm for dominating set analyzed by the potential method. In *IPEC (to appear)*, 2011.

[98] Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In Downey et al. [59], pages 235–247.

[99] Haim Kaplan, editor. *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, volume 6139 of *Lecture Notes in Computer Science*. Springer, 2010.

[100] K. Kawarabayashi and Y. Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the S-cycle packing problem (manuscript), 2010.

[101] Alexandr V. Kostochka. Lower bound of the hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984.

[102] Stefan Kratsch. Co-nondeterminism in compositions: A kernelization lower bound for a ramsey-type problem. *CoRR*, abs/1107.3704, 2011.

[103] Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, page to appear, 2012.

[104] R. Laskar and J. Pfaff. Domination and irredundance in graphs. Technical Report 434, Clemson Univ., Dept. of Math. SC., 1983.

[105] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.*, 19:544–546, 1973.

[106] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[107] J. K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[108] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *SODA*, pages 777–789. SIAM, 2011.

[109] François Margot, Maurice Queyranne, and Yaoguang Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.

[110] Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.

[111] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In Fortnow and Vadhan [80], pages 469–478.

[112] Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. In Md. Saidur Rahman and Satoshi Fujita, editors, *WALCOM*, volume 5942 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2010.

[113] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *FOCS*, pages 248–255. IEEE Computer Society, 2004.

[114] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006.

[115] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Charikar [29], pages 1065–1075.

[116] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In Fiat and Sanders [70], pages 694–705.

[117] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.

[118] Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009.

[119] Bruce A. Reed. Paths, stars and the number three. *Combinatorics, Probability & Computing*, 5:277–295, 1996.

[120] Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

[121] John H. Reif. Minimum s-t cut of a planar undirected network in $O(n\log^2(n))$ time. *SIAM J. Comput.*, 12(1):71–81, 1983.

[122] Ingo Schiermeyer. Efficiency in exponential time for domination-type problems. *Discrete Applied Mathematics*, 156(17):3291–3297, 2008.

[123] Alexander Schrijver. A short proof of Mader's sigma-paths theorem. *J. Comb. Theory, Ser. B*, 82(2):319–321, 2001.

[124] Andrew Thomason. An extremal function for contractions of graphs. *Math. Proc. Cambridge Philos. Soc.*, 95(2):261–265, 1984.

[125] Andrew Thomason. The extremal function for complete minors. *J. Comb. Theory, Ser. B*, 81(2):318–338, 2001.

[126] Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

[127] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Fiat and Sanders [70], pages 566–577.

[128] Vadim G. Vizing. Some unsolved problems in graph theory. *Uspehi Mat. Nauk*, 23(6 (144)):117–134, 1968.

[129] Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In Downey et al. [59], pages 281–290.

[130] Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.

[131] Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.