dc_1604_18

Tree decompositions of graphs and their effect on algorithmic complexity

Dániel Marx
Institute for Computer Science and Control,
Hungarian Academy of Sciences
(MTA SZTAKI)

Budapest
2019

Contents

CHAPTER 1

Introduction

A wide range of application areas require efficient algorithmic solutions for combinatorial problems. The theory of NP-hardness tells us that we cannot expect optimal polynomial-time algorithms for many of the optimization problems that arise in practice. While in these cases we cannot hope for efficient algorithms tackling the problem in its full generality, it may still be possible that an algorithm is provably efficient on certain special cases of practical or theoretical relevance. A large part of the theoretical research on algorithms can be seen as trying to identify favorable properties of the input instance that allow efficient algorithms.

Perhaps the most well-studied such property is the concept of *decomposability:* the problem instance can be recursively split into smaller parts such that the solutions of the smaller parts can be combined into a solution of the whole instance. The exact meaning of decomposability depends on the problem domain, but typically it means the existence of small separators that break the problem into fairly independent parts that only interact via the separator. For algorithmic problems that are explicitly defined on graphs (or perhaps there is a natural definition of a graph that can be associated with the problem instance), the notion of *treewidth* appears to be a very useful concept in formalizing decomposability. Informally speaking, treewidth measures how close a graph is to being a tree: graphs of small treewidth are similar to a tree with each node being replaced by a small graph. While the formal definition of treewidth is technical, it models very faithfully the requirements that make the algorithmic paradigm "split on small separators and recurse" work and its mathematical naturality is further evidenced by the fact that it was independently discovered in equivalent formulations at least three times [28, 135, 207].

The overarching theme of this dissertation is the quest for understanding how structural parameters such as treewidth and its variants make algorithmic problems provably more tractable. The results presented here include both algorithms (upper bounds on the running time) and complexity result (lower bounds on the running time). The results answer three different types of fundamental questions. First, given a certain algorithmic problem, we can ask if there is an efficient solution that works on instances that have low treewidth. In other words, we would like to know if treewidth helps resolving the problem in any way.

Does low value of treewidth (or a related measure) make it possible to solve certain type of problems efficiently?

After establishing that treewidth is a useful parameter, we would like to quantify this statement by understanding how exactly the running time can depend on treewidth. That is, we would like to have optimal algorithms that exploit treewidth as much as possible. Besides presenting the optimal algorithm, one needs to argue that the dependence on treewidth cannot be improved. Thus we need complexity results ruling out any possible improvement.

What is the best possible dependence on treewidth (or a related measure) that can be achieved for a given problem?

Finally, if we know exactly how treewidth influences the complexity of a problem, it is natural to ask if there is another graph property (perhaps some variant of treewidth) that decreases the complexity of the problem in a similar way.

Is bounded treewidth (or a related measure) the only graph-theoretical property that decreases the complexity of a given problem?

While questions of such form may sound very open ended, it is possible to define frameworks formalizing these questions and to give completely exhaustive answers [68, 69, 77, 123, 127, 154]. In particular, in some cases it is possible to prove "dichotomy theorems" showing formally that only the bounded-treewidth special cases are tractable, and *every other* special case is hard.

The goal of the rest of this introduction is twofold. It briefly goes over basic concepts that will be needed in the later chapters (such as treewidth and constraint satisfaction problems), serving as a shared prelimianaries section to avoid repeating the introduction of the same defininions all over again in each chapter. Moreover, we use this introduction to put the main results of the dissertation into context and briefly explain how they improve on what was known earlier. Here we highlight only a selection of results and state them in an informal manner; for formal statements of all the results, the reader is referred to the appropriate chapters.

Many of the results presented in the dissertation are complexity lower bounds. Of course, these results are all *conditional* lower bounds: as we cannot rule out P = NP at the moment, in principle it is possible that every problem considered here can be solved efficiently and these lower bounds are irrelevant. As it is customarily done in theoretical computer science, we assume a complexity hypothesis (such as P $\neq$ NP) and we prove the lower bounds under this assumption. Section 1.1 introduces these assumptions. The framework of parameterized complexity and fixed-parameter tractability is used explicitly or implicitly in most of the results; Section 1.2 gives a brief overview of these topics. Section 1.3 defines treewidth and introduces related graph-theoretical notions. Section 1.4 introduces one of the main application domains for our results: Constraint Satisfaction Problems (CSPs). Section 1.5 gives a brief overview of some basic notions of database theory relevant for our results. For reference, Section 1.6 summarizes the publication venues where the results presented in the different chapters of the dissertation originally appeared.

1.1 Complexity assumptions

The widely-believed complexity assumption P $\neq$ NP has immense explanatory power: via the theory of NP-completeness, this assumption explains why important combinatorial problems are not polynomial-time solvable. The use of the P $\neq$ NP hypothesis goes beyond explaining the hardness of finding optimal solutions. In recent decades, the field of inapproximability has seen enormous progress and very strong lower bounds on polynomial-time approximabilty can be obtained based on the P $\neq$ NP hypothesis [18, 86, 137].

However, there are important complexity lower bounds that currently are not known to be provable from P $\neq$ NP, requiring a stronger complexity assumption. For example, in the field of inapproximability, the Unique Games Conjectures introduced by Khot [162] allows proving tight lower bounds on the approximation ratio that match the best known algorithms [53, 131, 162, 163]. In the field of parameterized and exact computation, the P $\neq$ NP assumption only tells us that NP-hard problems do not admit polynomial-time algorithms, but does not say anything about the existence of subexponential-time algorithms. That is, it is still possible that there are much faster algorithms for NP-hard problems than what we know today: for example, as far as we know, the P $\neq$ NP assumption does not rule out algorithms with running time $2^{n/10000}$, or $2^{n^{1/100}}$, or $n^{\log n}$, or even $n^{\log\log n}$ for an NP-hard problem such as 3SAT with n variables.

The Exponential-Time Hypothesis (ETH), formulated by Impagliazzo, Paturi, and Zane [145,146], makes the assumption P $\neq$ NP more quantitative: informally, it not only tells us that NP-hard problems do not have polynomial-time algorithms, but it posits that NP-hard problems really require exponential time and cannot be solved in subexponential time. The formal statement of the ETH is somewhat technical and for most applications it is more convenient to use the following assumption instead, which is an easy consequence of the ETH:

Hypothesis 1.1 (Consequence of the ETH, Impagliazzo, Paturi, and Zane [145, 146])**.** 3SAT *with n variables cannot be solved in time $2^{o(n)}$.*

3SAT is the fundamental satisfiability problem where, given a Boolean formula in conjunctive normal form with at most 3 literals in each clase (e.g., $(x_1 \vee \bar{x}_3 \vee x_5) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$), the task is to decide whether a satisfying assignment exists. For completeness, let us recall the formal statement of the ETH, of which Hypothesis 1.1 is an easy consquence. Let s_k be the infinum of all real numbers δ for which there exists an $O(2^{\delta n})$ time algorithm for k-SAT. Then the ETH is the assumption that $s_k > 0$ for every $k \geq 3$. It is easy to show that this assumption implies Hypothesis 1.1, hence if we can show that some statement would refute Hypothesis 1.1, then it would refute the ETH as well.

Hypothesis 1.1 rules out the existence of algorithms that are subexponential in the number n of variables. But the number m of clauses in a 3SAT instance can be up to cubic in the number of variables, thus the length of the instance can be much larger than $O(n)$. Therefore, Hypothesis 1.1 does not rule out the existence of algorithms that are subexponential in the length of the instance: it could be potentially the case that all the really hard instances of 3SAT have, say, $\Omega(n^2)$ clauses, hence a $2^{o(\sqrt{m})}$ algorithm would be still compatible with Hypothesis 1.1. Impagliazzo, Paturi and Zane [146] showed that this is not the case: the Sparsification Lemma implies that, for the purposes of Hypothesis 1.1, 3SAT remains hard already when restricted to instances with a linear number of clauses. With the Sparsification Lemma, the following stronger assumption follows from Hypothesis 1.1:

Hypothesis 1.2 (Consequence of the ETH + Sparsification Lemma, Impagliazzo, Paturi, and Zane [146])**.** 3SAT *with n variables and m clauses cannot be solved in time $2^{o(n+m)}$.*

This stronger assumption turns out to be very useful to prove lower bounds for other problems. Reductions from 3SAT to other problems typically create instances whose size depend not only on the number n of variables, but also on the number m of clauses, hence it is important to have lower bounds on 3SAT in terms of both n and m.

Despite the usefulness of the ETH, there are complexity lower bounds that seem to be beyond the reach of what can be proved as a consequence of this hypothesis. Impagliazzo, Paturi, and Zane [146] proposed an even stronger assumption on the complexity of NP-hard problems: the so-called Strong Exponential-Time Hypothesis (SETH). Using the notation introduced above, the SETH assumes that $\lim_{k\to\infty} s_k = 1$. The following consequence of the SETH is a convenient formulation that can be used as a starting point for lower bounds on other problems:

Hypothesis 1.3 (Consequence of the SETH, Impagliazzo, Paturi, and Zane [146])**.** SAT *with n variables and m clauses cannot be solved in time $(2-\epsilon)^n \cdot m^{O(1)}$ for any $\epsilon > 0$.*

Intuitively, Hypothesis 1.3 states that there is no better algorithm for SAT than the brute force search of trying each of the 2^n possible assignments. Note that here SAT is the satisfiability problem with unbounded clause length. For fixed clause length, algorithms better than 2^n are known: for example, the best known algorithms for 3SAT and 4SAT have running times 1.308^n and 1.469^n, respectively [138]. The SETH states that the base of the exponent has to get closer and closer to 1 as the clause length increases, and it is not possible to have an algorithm with base $2-\epsilon$ that works for arbitrary large clause length.

It is important to note that there is no known analogue of the Sparsification Lemma for the SETH. That is, we cannot assume that the hard instances stipulated by Hypothesis 1.3 have only a linear number of clauses: for all we know, the number of clauses can be exponential in the number n of variables. This severly limits the applicability of lower bounds based on the SETH as any reduction from the SAT instance would create instances whose sizes are potentially exponentially large in n. Nevertheless, the SETH has found applications in parameterized complexity, where instead of giving a lower bound on how the running time has to depend on the instance or solution size, we want to understand how it depends on some other parameter. Chapter 2 contains a selection of such results, giving tight lower bounds on how the running time has to depend on treewidth. The results presented in this chapter were the first such tight lower bounds on parameterization by treewidth, inspiring many subsequent results of this form [38, 70, 72, 76, 91, 151, 152]. In recent years, the SETH has been successfully used to give lower bounds for polynomial-time solvable problems, for example, by showing that the textbook $O(n^2)$ dynamic programming algorithm for Edit Distance cannot be significantly improved: it cannot be solved in time $O(n^{2-\epsilon})$ for any $\epsilon > 0$, unless the SETH fails [22, 43]. Many other tight results of this form can be found in the recent literature under the name "fine-grained complexity" [1–3, 22, 39, 40, 43, 44, 202, 210, 232].

The ETH and the SETH are stronger assumptions that the P $\neq$ NP hypothesis and perhaps not as widely accepted in the research community. Thus one may wonder about the meaning of conditional lower bounds based on them. First, by now the ETH is fairly well accepted in the research community and has been used as the starting point to prove lower bounds for numerous problems (e.g., [16,73,80,104,106,175]). Indeed, despite decades of research on 3SAT, there is no sign whatsoever that subexponential algorithms would be possible and in fact minor improvements of the base of the exponential function in the running time required considerable efforts [138, 139, 148, 170, 173, 193]. The SETH is a much more ambitious proposition and there is less evidence supporting its validity. Still, the explanatory power of this hypothesis has been used in a large number of recent research results and obtaining conditional lower bounds based on it has become a mainstream research direction. Regardless of what the reader may think of the validity of the SETH, the following point of view should clarify why results of this form are still valuable. A conditional lower bound showing

that "a better algorithm for problem X would violate the SETH" shows that in order to improve the algorithm for problem X, one needs to be able to deal with the SAT problem in its full generality. It is not the particular difficulties of problem X that needs to be better understood to improve the running time: improvements are prevented by the lack of better understanding of satisfiability itself. Thus it can be argued that instead of focusing on problem X, one should better focus on the more fundamental question of improving SAT algorithms directly, because this is what improvements on problem X would eventually achieve. In other words, the conditional lower bound closes the question "Are there better algorithms for problem X?" by saying one should not directly work on this question until the validity of the SETH is resolved one way or the other.

1.2 Parameterized complexity

Classical complexity theory expresses the running time of an algorithm as a univariate function of the size n of the input. The main conceptual idea of parameterized complexity is to introduce additional parameters and to express the running time as a multivariate function of the input size and these parameters. The goal is to find algorithms where the exponential growth of the running time is restricted to these parameters, while the running time depends only polynomially on the size of the input. If we have an application where the parameters can be assumed to be small, then such an algorithm can be efficient even for large input sizes.

Formally, we associate an integer parameter k with each input instance of the problem and we say that the problem is *fixed-parameter tractable (FPT)* if the problem can be solved in time $f(k)n^c$, where f is an arbitrary computable function depending only on k and c is a constant (independent of k). If the function $f(k)$ is a moderately growing exponential function and d is not too large, then an FPT-time algorithm can be efficient for applications where k is small. For example, an algorithm with running time, say, $O(1.2852^k + kn)$ for k-Vertex Cover [61] is feasible for $k = 40$, since in this case the term 1.2852^k is only about 22000.

The classical work of Downey and Fellows [87] summarized the results of the field up to 1999, with more recent monographs [74, 88, 102, 197] providing an introduction to the enormous progress that happened since then. In the past 20 years, the parameterized complexity community identified hundreds of NP-hard problems that are fixed-parameter tractable with natural parameterizations. The most studied graph-theoretic examples include finding a cycle of length exactly k [13, 165, 231], finding a vertex cover of size k [5,61,62,82,129,220], finding k vertex disjoint triangles in a graph [169], and various problems parameterized by the treewidth k of the input graph (cf. [17, 32]). Systematic search efforts were undertaken to find fixed-parameter tractable problems in various problem domains such as artificial intelligence [118], computational biology [50, 120, 182], and geometric problems [111, 181, 191]. The first results were obtained by ad hoc techniques, but (especially in the past decade) there have been intensive efforts to understand the methods used for obtaining FPT-time algorithms, turning them into general techniques. Currently, we have an impressive toolbox of algorithmic techniques at our disposal (cf. [74, 144, 218, 219]). However, many problems resisted all algorithmic techniques and no FPT-time algorithm is known for them. The theory of W[1]-hardness is the parameterized-complexity analog of NP-hardness, and can be used to give strong theoretical evidence that a problem is not fixed-parameter tractable. Although technically more difficult to prove than NP-hardness, the W[1]-hardness of numerous problems has been shown in the literature. Intuitively, Clique is W[1]-complete and W[1]-hard means that it is at least as hard as Clique from the viewpoint of fixed-parameter tractability. As the current dissertation does not present any W[1]-hardness proofs (all the lower bounds are based on the ETH or the SETH), the formal definition of W[1]-hardness will not be important.

The parameter k of the instance can be any well-defined measure of the instance. For a given algorithmic problem, one can define different natural parameters, leading to different parameterizations of the same problem. It may very well be the case that the same problem is fixed-parameter tractable with one parameterization and W[1]-hard with some other. For optimization problems (for example, finding a clique, dominating set, path etc. of maximum/minimum size), the most natural parameterization is by the size of the solution we are looking for. That is, we assume that the input instance contains a target number k, and the task is to find a solution of size at least/at most k in time $f(k)n^{O(1)}$. But there are many other potential parameters one can define. The input may already contain different values that one can choose as the parameter: for example, Partial Vertex Cover asks for the selection of k vertices that cover at least ℓ edges — one can parameterize by either k or ℓ in this problem. The parameter can be a measure of some aspect of the input instance: the maximum degree or treewidth in a graph problem, the dimension of the point set in a geometric problem, the alphabet size or the length of the strings in a pattern matching problem, or the domain size of the variables in a constraint satisfaction problem. Even for a single problem, one can discover a rich and nontrivial complexity landscape by considering different (combinations of) parameters [41, 189].

1.3 Graphs and hypergraphs

A large part of the dissertation deals with graphs and graph-like structures. We provide a brief overview of the main notions here, including the definition of treewidth, which will be used in many of the chapters. For ease of readability, some of the definitions will be repeated later. For more background on graph theory, the reader is referred to, e.g., the text book of Diestel [85].

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For a subset $V' \subseteq V(G)$, the subgraph $G' = G[V']$ of G is called a *subgraph induced by* V' if $E(G') = \{uv \in E(G) \mid u, v \in V'\}$ and $V(G') = V'$. By $N(u)$ we denote the (open) neighborhood of u in graph G, that is, the set of all vertices adjacent to u and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V(G)$, we define $N[D] = \cup_{v \in D} N[v]$.

Treewidth. A *tree decomposition* of a graph G is a pair $(\mathcal{B}, T)$ where T is a tree and $\mathcal{B} = \{B_t \mid t \in V(T)\}$ is a collection of subsets of $V(G)$ such that:

- $\bigcup_{t \in V(T)} B_t = V(G)$,
- for each edge $xy \in E(G)$, $\{x, y\} \subseteq B_t$ for some $t \in V(T)$;
- for each $x \in V(G)$ the set $\{t \mid x \in B_t\}$ induces a connected subtree of T.

The sets B_t are called the *bags* of the decomposition. Sometimes it will be convenient for us to view the trees in a tree-decomposition as being rooted and directed from the root to the leaves. For a node t in a (rooted) tree $T = (V(T), E(T))$, we let T_t be the subtree rooted at t, that is, the induced subtree of T whose vertex set is the set of all vertices reachable from t.

The *width* of the tree decomposition is $\max_{t \in V(T)}\{|B_t| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We denote by $\mathrm{tw}(G)$ the treewidth of graph G. It is known that $\mathrm{tw}(G) \leq 1$ if and only if G is a forest (has no cycles). If in the definition of treewidth we restrict the tree T to be a path then we get the notion of *pathwidth* and denote it by $\mathrm{pw}(G)$. By definition, we have that $\mathrm{tw}(G) \leq \mathrm{pw}(G)$, but it is known that pathwidth cannot be bounded by any function of treewidth: if G is a complete binary tree width $2k + 1$ levels, then $\mathrm{tw}(G) = 1$ and $\mathrm{pw}(G) = k$.

We say that a class $\mathcal{G}$ of graphs is of *bounded treewidth* if there is a k such that $\mathrm{tw}(G) \leq k$ for all $G \in \mathcal{G}$. We use a similar terminology for other graph invariants.

Computing the treewidth of a given graph is known to be NP-hard [17], but for every fixed $k \geq 1$, there is a linear-time algorithm for finding a decomposition of width k, if exists [31]. That is, finding a decomposition of width k is fixed-parameter tractable. In polynomial time, one can compute a tree decomposition of width $O(\mathrm{tw}(G) \cdot \sqrt{\mathrm{tw}(G)})$ [15].

There are well-known classes of graphs that have small treewidth, for example series-parallel and outerplanar graphs have treewidth at most 2, and every planar graph on n vertices has treewidth $O(\sqrt{n})$ [30,33]. One can argue that real-word graphs that exhibit some hierarchical structure should have treewidth much smaller than the number of vertices [132,178,226]. For such graphs, algorithms exploiting tree decompositions can explain the tractability of various problems.

Given a tree decomposition of width k, the standard technique of dynamic programming over a tree decomposition can result in algorithms with running time $f(k) \cdot n$ for various fundamental combinatorial problems such as Independent Set, Vertex Cover, Dominating Set, Hamiltonian Cycle, 3-Coloring, etc. Even though the process of designing such a dynamic programming algorithm is fairly standard, one still needs to define partial subproblems in a problem-specific way, sometimes resulting in long and cumbersome proofs. In many cases, Courcelle's Theorem [67] automatizes this process: as a powerful and far-reaching generalization of simple dynamic programming algorithms, it shows that *every* decision problem that can be described as a logical formula ϕ in Extended Monadic Second Order Logic can be solved in time $f(\phi, k) \cdot n$ on graphs of treewidth at most k.

While Courcelle's Theorem immediatelly gives an $f(k) \cdot n$ time algorithm on graphs of treewidth k for, say, 3-Coloring, it does not give good bounds on the growth rate of the function $f(k)$. Often, problem-specific techniques can deliver algorithms with better bounds on $f(k)$. While for Independent Set it is fairly straightforward to obtain an algorithm with running time $2^k \cdot n^{O(1)}$, the $3^k \cdot n^{O(1)}$ time algorithms for Dominating Set required the application of the nontrivial Fast Subset Convolution technique [29,227] and the first $c^k \cdot n^{O(1)}$ time algorithms for Hamiltonian Cycle was made possible only with the *Cut & Count* technique [76] and the rank-based approaches [34,75].

Given that for some problems new techniques were necessary to improve the dependence on treewidth and for others essentially no progress was made, it is natural to ask what the best possible dependence $f(k)$ on treewidth is. The results in Chapter 2 answer precisely this question: they show that, assuming the SETH, for several basic problems (such as Indpendent Set and 3-Coloring) the best known algorithm has essentially optimal dependence on treewidth, up to an ϵ term in the base of the exponential function. These results closed the question of whether further improvements can be expected for these problems. At the time of publication of the conference version of this work [174] in SODA 2011, these results were the first results proving such tight lower bounds for parameterization by treewidth; inspired by this work, similar results were published later by other authors [38,70,72,76,91,151,152].

Hypergraphs. In a graph, each edge connects two vertices, representing a relation of arity two such as "connected", "adjacent", or "conflicts with," etc. In some cases, the natural representation of a problem requires expressing relations of higher arity, for example, expressing that certain items appear in groups. *Hypergraphs* are natural generalizations of graphs, allowing hyperedges containing more than two vertices. Formally, a *hypergraph* is a pair $H = (V(H), E(H))$, consisting of a set $V(H)$ of *vertices* and a set $E(H)$ of nonempty subsets of $V(H)$, the *hyperedges* of H.

For a hypergraph H and a set $X \subseteq V(H)$, the *subhypergraph of H induced by X* is the hypergraph $H[X] = (X, \{e \cap X \mid e \in E(H) \text{ with } e \cap X \neq \emptyset\})$. We let $H \setminus X = H[V(H) \setminus X]$. The *primal graph*

of a hypergraph H is the graph

$$\underline{H} = (V(H), \{\{v,w\} \mid v \neq w, \text{ there exists an } e \in E(H) \text{ such that } \{v,w\} \subseteq e\}).$$

A hypergraph H is *connected* if $\underline{H}$ is connected. A set $C \subseteq V(H)$ is *connected (in H)* if the induced subhypergraph $H[C]$ is connected, and a *connected component* of H is a maximal connected subset of $V(H)$. A sequence of vertices of H is a *path* of H if it is a path of $\underline{H}$.

Let us generalize the notion of tree decomposition and treewidth to hypergraphs. A *tree decomposition* of a hypergraph H is a pair $(\mathcal{B}, T)$ where T is a tree and $\mathcal{B} = \{B_t \mid t \in V(T)\}$ is a collection of subsets of $V(H)$ such that:

- $\bigcup_{t \in V(T)} B_t = V(H)$,
- for each hyperedge $e \in E(H)$, we have $e \subseteq B_t$ for some $t \in V(T)$;
- for each $x \in V(H)$ the set $\{t \mid x \in B_t\}$ induces a connected subtree of T.

The sets B_t are called the *bags* of the decomposition. The *width* of a tree-decomposition $(T, \mathcal{B})$ is $\max\{|B_t| \mid t \in V(T)\} - 1$. The *treewidth* $\mathrm{tw}(H)$ of a hypergraph H is the minimum of the widths of all tree-decompositions of H. It is well-known and easy to see that $\mathrm{tw}(H) = \mathrm{tw}(\underline{H})$ for all H.

1.4 Constraint Satisfaction Problems

Constraint satisfaction is a general framework that includes many standard algorithmic problems such as satisfiability, graph coloring, database queries, etc. A constraint satisfaction problem (CSP) consists of a set V of variables, a domain D, and a set C of constraints, where each constraint is a relation on a subset of the variables. The task is to assign a value from D to each variable in such a way that every constraint is satisfied (see Definition 1.4 below for the formal definition). For example, 3SAT can be interpreted as a CSP instance where the domain is $\{0,1\}$ and the constraints in C correspond to the clauses (thus the arity of each constraint is 3). Another example is vertex coloring, which can be interpreted as a CSP instance where the variables correspond to the vertices, the domain corresponds to the set of colors, and there is a binary "not equal" constraint corresponding to each edge. Notice that the domain size can be arbitrarily large in the CSP instances arising from vertex coloring (as the coloring problem might involve any number of colors). In the this dissertation, we think of the domain as a set whose size is not a fixed constant, but can be be arbitrarily large. This viewpoint is natural in the context of various database query and artificial intelligence applications, where in fact that domain size is usually much larger than the number of variables [118, 211].

Basic definitions. We briefly recall some terminology related to CSP. For more background, see, for example, [96, 122].

Definition 1.4. An instance I of a *constraint satisfaction problem* is a triple $I = (V, D, C)$, where:

- V is a set of variables,
- D is a domain of values,
- C is a set of constraints, $\{c_1, c_2, \ldots, c_q\}$. Each constraint $c_i \in C$ is a pair $\langle s_i, R_i \rangle$, where:
  - s_i is a tuple of variables of length m_i, called the *constraint scope,* and
  - R_i is an m_i-ary relation over D, called the *constraint relation.*

For each constraint $\langle s_i, R_i \rangle$ the tuples of R_i indicate the allowed combinations of simultaneous values for the variables in s_i. The length m_i of the tuple s_i is called the *arity* of the constraint. We allow repeated variables in the scope s_i, but this does not make the problem more general and can be usually ignored. A *solution* to a constraint satisfaction problem instance is a function f from the set of variables V to the domain D of values such that for each constraint $\langle s_i, R_i \rangle$ with $s_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_m})$, the tuple $(f(v_{i_1}), f(v_{i_2}), \ldots, f(v_{i_m}))$ is a member of R_i. In the decision version of CSP, we have to decide if a solution for the given instance I exists. Observe that there is a polynomial-time algorithm deciding whether a given assignment for an instance is a solution.

The *primal graph* (or *Gaifmann graph*) of a CSP instance $I = (V, D, C)$ is a graph G with vertex set V, where $x, y \in V$ form an edge if and only if there is a constraint $\langle s_i, R_i \rangle \in C$ with $x, y \in s_i$. For a class $\mathcal{G}$ of graphs, we denote by $\mathrm{CSP}(\mathcal{G})$ the problem restricted to instances where the primal graph is in $\mathcal{G}$. Note that this definition does not make any restriction on the constraint relations: it is possible that every constraint has a different constraint relation.

The *hypergraph* of an instance $I = (V, D, C)$ has V as its vertex set and for every constraint in C a hyperedge that consists of all variables occurring in the constraint. For a class $\mathcal{H}$ of hypergraphs, we let $\mathrm{CSP}(\mathcal{H})$ be the class of all instances whose hypergraph is contained in $\mathcal{H}$.

We say that an instance is *binary* if each constraint relation is binary, that is, $m_i = 2$ for every constraint[1]. It can be assumed that the instance does not contain two constraints $\langle s_i, R_i \rangle$, $\langle s_j, R_j \rangle$ with $s_i = s_j$, since in this case the two constraints can be replaced by the constraint $\langle s_i, R_i \cap R_j \rangle$.

Representation of the constraints. In the input, the relation in a constraint is represented by listing all the tuples of the constraint. We denote by $\|I\|$ the size of the representation of the instance $I = (V, D, C)$. For binary constraint satisfaction problems, we may assume that $\|I\| = O(V^2 D^2)$; by the argument in the previous paragraph, we may assume that there are $O(V^2)$ constraints and each constraint has a representation of length $O(D^2)$. Furthermore, it can be assumed that $|D| \leq \|I\|$; elements of D that do not appear in any relation can be removed.

If constraints of larger arity are also allowed in the input, we have to be more careful and precise in describing how the constraints are represented and how this representation contributes to the input size. Throughout this dissertation, we assume that the constraints are specified by explicitly enumerating all possible combinations of values for the variables, that is, all tuples in the relation R. Consequently, we define the *size* of a constraint $c = \langle (v_1, \ldots, v_k), R \rangle \in C$ to be the number $\|c\| = k + k \cdot |R|$. The *size* of an instance $I = (V, D, C)$ is the number $\|I\| = |V| + |D| + \sum_{c \in C} \|c\|$. Of course, there is no need to store a constraint relation repeatedly if it occurs in several constraints, but this only changes the size by a polynomial factor.

Let us make a few remarks about this explicit representation of the constraints. There are important special cases of constraint satisfaction problems where the constraints are stored implicitly, which may make the representation exponentially more succinct. Examples include Boolean satisfiability, where the constraint relations are given implicitly by the clauses of a formula in conjunctive normal form, or systems of arithmetic (in)equalities, where the constraints are given implicitly by the (in)equalities. However, our representation is the standard "generic" representation of constraint satisfaction problems in artificial intelligence (see, for example, [81]). An important application where the constraints are always given in explicit form is the conjunctive query containment problem, which plays a crucial role in database query optimization. Kolaitis and Vardi [166] observed that it can be represented as a constraint satisfaction problem, and the constraint relations are given explicitly as part of one of the input queries. A related problem from database systems is the problem of evaluating conjunctive queries (see Section 1.5 below). Here the constraint relations represent the

[1]It is unfortunate that while some communities use the term "binary CSP" in the sense that each constraint is binary (as does this dissertation), others use it in the sense that the variables are 0-1, that is, the domain size is 2.

tables of a relational database, and again they are given in explicit form. The problem of characterizing the tractable structural restrictions of CSP has also been studied for other representations of the instances: one can consider more succinct representations such as disjunctive formulas or decision diagrams [58] or less succinct representations such as truth tables [187]. As the choice of representation influences the size of the input and the running time is expressed as a function of the input size, the choice of representation influences the complexity of the problem and the exact tractability criterion.

Complexity classifications. Due to its generality, solving constraint satisfaction problems is NP-hard if we do not impose any additional restrictions on the possible instances. Therefore, the main goal of the research on CSP is to identify tractable classes and special cases of the general problem. The theoretical literature on CSP investigates two main types of restrictions. The first type is to restrict the *constraint language,* that is, the type of constraints that is allowed. This direction was initiated by the classical work of Schaefer [212] and was subsequently pursued in, e.g., [45,46,49,96,156]. Recently, as a major breakthrough, Bulatov [47] and Zhuk [234] independently characterized the complexity of every CSP problem with a fixed constraint language, resolving a long-standing open problem raised by Feder and Vardi [96]. Significant progress was made on the complexity of the optimization versions of the problem as well [79, 168, 224].

The second type is to restrict the *structure* induced by the constraints on the variables. The goal is to understand what structural properties of the CSP instance can make the problem easier. The first question is to understand which graphs make CSP polynomial-time solvable. We have to be careful with the formalization of this question: if G is a graph with k vertices, then any CSP instance with primal graph G can be solved in time $n^{O(k)}$ by brute force. Therefore, restricting CSP to *any* fixed graph G makes it polynomial-time solvable. The real question is which *classes* of graphs make the problem polynomial-time solvable: using the definitions introduced above, which classes $\mathcal{G}$ of graphs make CSP($\mathcal{G}$) polynomial-time solvable? Freuder [108] observed that if the treewidth of the primal graph is k, then CSP can be solved in time $n^{O(k)}$. Thus if $\mathcal{G}$ has bounded treewidth, then CSP($\mathcal{G}$) is polynomial-time solvable. Quite surprisingly, the converse is also known to be true: it follows from the work of Grohe, Schwentick, and Segoufin [123, 127] that whenever $\mathcal{G}$ is *any* recursively enumerable class of graphs with unbounded treewidth, then CSP($\mathcal{G}$) is *not* polynomial-time solvable, unless FPT = W[1].

By this result of Grohe, Schwentick, and Segoufin, bounded treewidth is the only property of the primal graph that can make the problem polynomial-time solvable. However it does not rule out the possibility that there is some structural property that may enable us to solve instances significantly faster than the treewidth-based algorithm of [108], that is, for some class $\mathcal{G}$ of graphs with unbounded treewidth, CSP($\mathcal{G}$) could be solved in time $n^{f(k)}$ where k is the treewidth of the primal graph and f is a slowly growing function such as $\sqrt{k}$ or $\log k$. The main result of Chapter 3 is that this is not possible; the $n^{O(k)}$-time algorithm is essentially optimal for every class of graphs, up to an $O(\log k)$ factor in the exponent. It follows as consequence of this result is that, assuming the ETH, there is no $f(k)n^{o(k/\log k)}$ time algorithm for the graph-theoretic problem Partitioned Subgraph Isomorphism, where k is the number of edges of the pattern to be found and f is an arbitrary computable function. This lower bound turned out be a very useful starting point for proving almost tight lower bound of this form for several other W[1]-hard parameterized problems in different domains. As there is no other technique currently that would give bounds of such tightness, its use has become a standard techique that was invoked several times (e.g., [35–37, 42, 68, 71, 92, 94, 128, 155, 158, 177, 190, 203]).

Large arities. For binary CSP instances, the primal graph G completely describes the structure induced by the constraints. But if there are constraints of higher arity, then the primal graph loses information. For example, by looking at the primal graph only, we cannot tell whether the instance contains $\binom{k}{2}$ binary constraints on k variables or just a single k-ary constriant on all the variables:

the primal graph is a complete graph on k vertices in both cases. Thus the hypergraph H of the CSP instance contains more information than the primal graph G, hence classifying the complexity of CSP($\mathcal{H}$) for every class $\mathcal{H}$ of hypergraphs gives a more refined classification than classifying CSP($\mathcal{G}$) for every class $\mathcal{G}$ of graphs.

If a class $\mathcal{H}$ of hypergraphs has bounded arity (i.e., there is an integer c such that every edge of every hypergraph in $\mathcal{H}$ has size at most c), then the classification for binary CSPs essentially goes through, and it follows that bounded treewidth of the hypergraph class $\mathcal{H}$ is the only property that makes the problem polynomial-time solvable. However, the situation significantly changes if $\mathcal{H}$ has unbounded arity. Consider for example the class $\mathcal{H}_1$ containing every hypergraph where there is an edge that covers every vertex. Now the primal graph of each $H \in \mathcal{H}_1$ is a complete graph, thus $\mathcal{H}_1$ does not have bounded treewidth. But CSP($\mathcal{H}_1$) is polynomial-time solvable: as every hypergraph in $\mathcal{H}_1$ contains an edge covering every vertex, every instance of CSP($\mathcal{H}_1$) contains a constraint c involving every variable. Thus all we need is going through the satisfying assignments of constraint c and check whether one of them satisfies every other constraint (recall that we have assumed above that constraints are represented in the input by explicitly listing every assignment that satisfies it). Thus bounded treewidth of the hypergraph class $\mathcal{H}$ is not the right tractability criterion for characterizing the polynomial-time solvable cases of CSP($\mathcal{H}$) and the situation seems to be much more complicated. As a side note, one could say that the tractability of CSP($\mathcal{H}_1$) is only an artifact of the assumption that the constraints are represented by listing every satisfying assignment. However, as we shall see in Section 1.5, this assumption is very natural in the context of database queries, and the original motivation for this line of research was studying applications in database theory.

Generalizing the simple example of $\mathcal{H}_1$ in the previous paragraph, it was shown that there is a notion of acyclicity for hypergraphs that makes the problem polynomial-time solvable [27, 95, 233]. Generalizing acyclicity, Gottlob et al. [115–117] introduced the notion of *hypertree width* and showed that CSP($\mathcal{H}$) is polynomial-time solvable if $\mathcal{H}$ has bounded hypertree width. Adler et al. [8] introduced *generalized hypertree width,* but this notion does not deliver new polynomial-time solvable cases of CSP($\mathcal{H}$): it is known that $\mathcal{H}$ has bounded hypertree width if and only if it has bounded generalized hypertree width.

Chapter 4 introduces a the notion of *bounded fractional edge cover number* and shows that if $\mathcal{H}$ has this property, then CSP($\mathcal{H}$) is polynomial-time solvable. As this property is incomparable to bounded hypertree width, it gives new tractable classes that were not known before. Chapter 5 introduces *fractional hypertree width* as a common generalization of fractional edge covers and hypertree width. It is shown that if $\mathcal{H}$ has bounded fractional hypertree width, then CSP($\mathcal{H}$) is polynomial-time solvable, making this property the currently known most general structural property that makes CSP($\mathcal{H}$) polynomial-time solvable. Figure 1.1 shows some of the known tractable hypergraph properties (note that the elements of this Venn diagram are sets of hypergraphs; e.g., the set "bounded treewidth" contains every set $\mathcal{H}$ of hypergraphs with bounded treewidth). All the inclusions in the figure are known to be proper.

Currently it is not known if there is any hypergraph class $\mathcal{H}$ with unbounded fractional hypertree width that make CSP($\mathcal{H}$) polynomial-time solvable. However, if we consider a weaker form of tractability, then we can obtain algorithms for some classes with unbound fractional hypertree width. Instead of asking for a polynomial-time algorithm, we can ask if the problem is fixed-parameter tractable parameterized by the number k of variables, that is, there is an algorithm with running time $f(k)n^{O(1)}$ for some function f. This question is very natural in settings where the number of variables is small, but the domain size is large, making the size of the relations and the total input size much larger than the number of variables. Chapter 6 introduces the notion of *submodular width* and shows that if $\mathcal{H}$ has is a class of hypergraphs with bounded submodular width, then CSP($\mathcal{H}$) is

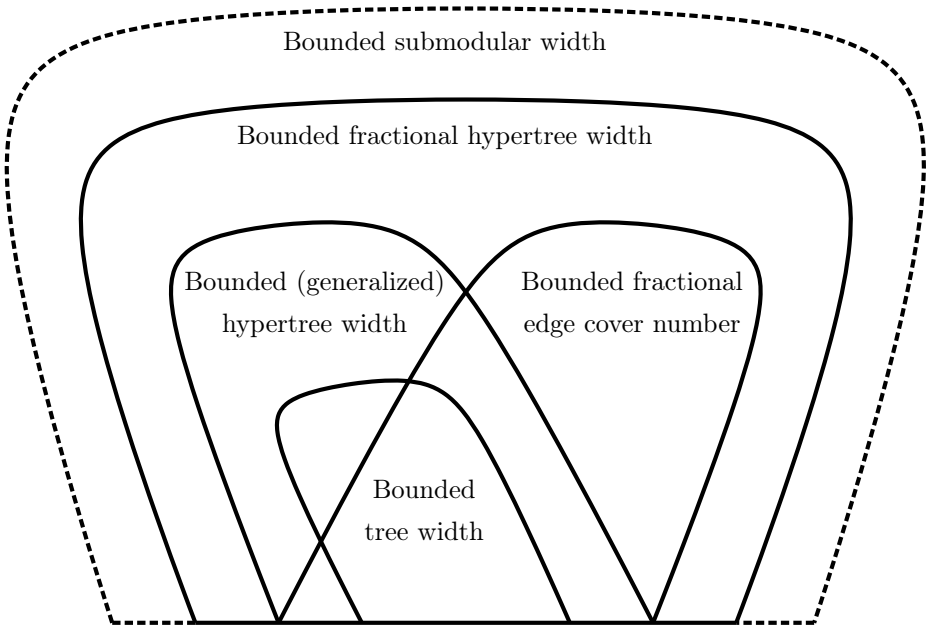

Figure 1.1: Hypergraph properties that make CSP tractable. Bounded submodular width makes the problem fixed-parameter tractable, while all the othe properties make the problem polynomial-time solvable.

fixed-parameter tractable parameterized by the number k of variables, with a function $f(k)$ in the running time that is double-exponential in k. Conversely, Chapter 6 shows that if $\mathcal{H}$ has unbounded submodular width, then CSP($\mathcal{H}$) is not fixed-parameter tractable parameterized by the number of variables, unless the ETH fails. Thus we get a complete characterization of hypergraph classes that make CSP fixed-pararameter tractable.

1.5 Database queries

Evaluation of conjunctive queries (or equivalently, Select-Project-Join queries) is one of the most basic and most studied tasks in relational databases. A relational database consists of a fixed set of relations. A conjunctive query defines a new relation that can be obtained as first taking the join of some relations and then projecting it to a subset of the variables. As an example, consider a relational database that contains three relations: enrolled(Person, Course, Date), teaches(Person, Course, Year), parent(Person1, Person2). The following query Q defines a unary relation ans(P) with the meaning that "P is enrolled in a course taught by her parent."

$$Q : \text{ans}(P) \leftarrow \text{enrolled}(P, C, D) \wedge \text{teaches}(P2, C, Y) \wedge \text{parent}(P2, P).$$

In the *Boolean Conjunctive Query* problem, the task is only to decide if the answer relation is empty or not, that is, if the join of the relations is empty or not. This is usually denoted as the relation "ans" not having any variables. Boolean Conjunctive Query contains most of the combinatorial difficulty

of the general problem without complications such that the size of the output being exponentially large. Therefore, the current dissertation focuses on this decision problem.

In a natural way, we can define the *hypergraph* of a query: its vertices are the variables appearing in the query and for each relation there is a corresponding hyperedge containing the variables appearing in the relation. Intuitively, if the hypergraph has "simple structure," then the query is easy to solve. For example, compare the following two queries:

$$Q_1 : \text{ans} \leftarrow R_1(A, B, C) \wedge R_2(C, D) \wedge R_3(D, E, F) \wedge R_4(E, F, G, H) \wedge R_5(H, I)$$
$$Q_2 : \text{ans} \leftarrow R_1(A, B) \wedge R_2(A, C) \wedge R_3(A, D) \wedge R_4(B, C) \wedge R_5(B, D) \wedge R_6(C, D)$$

Even though more variables appear in Q_1, evaluating it seems to be easier: its hypergraph is "path like," thus the query can be answered efficiently by, say, dynamic programming techniques. On the other hand, the hypergraph of Q_2 is a clique on 4 vertices and no significant shortcut is apparent compared to trying all possible combinations of values for (A, B, C, D).

What are those hypergraph properties that make Boolean Conjunctive Query tractable? In the early 80s, it has been noted that acyclicity is one such property [25, 26, 95, 233]. Later, more general such properties were identified in the literature: for example, bounded query width [56], bounded hypertree width [116], and bounded fractional hypertree width [126, 184]. Our goal is to find the most general hypergraph property that guarantees an efficient solution for query evaluation.

It is easy to see that Boolean Conjunctive Query can be formulated as the problem of deciding if a CSP instance has a solution: the variables of the CSP instance correspond to the variables appearing in the query and the constraints correspond to the database relations. A distinctive feature of CSP instances obtained this way is that the number of variables is small (as queries are typically small), while the domain of the variables are large (as the database relations usually contain a large number of entries). This has to be contrasted with typical CSP problems from AI, such as 3-colorability and satisfiability, where the domain is small, but the number of variables is large.

As discussed in Section 1.4, we assume that the constraints in a CSP instance are represented by explicitly listing all tuples that satisfy the constraint. This representation is perfectly compatible with our database-theoretic motivation: the constraints are relations of the database, and a relation is physically stored as a table containing all the tuples in the relation. For this representation, there are classes $\mathcal{H}$ with unbounded treewidth such that CSP restricted to this class is polynomial-time solvable. As mentioned in Section 1.4, with this representation CSP($\mathcal{H}$) is polynomial-time solvable whenever $\mathcal{H}$ has bounded fractional hypertree width and CSP($\mathcal{H}$) is fixed-parameter tractable parameterized by the number of variables whenever $\mathcal{H}$ has bounded submodular width.

Instead of just deciding if the answer to the query is empty or not (the Boolean Conjunctive Query Problem), one may want to enumerate every solution. Efficient enumeration of every solution is not always possible, simply because the number of solutions can be exponentially large even for very simple queries. Therefore, typical goals for enumeration is to obtain a running time that is constant or polynomial per the number of solutions, or more restrictively, there is only a contant-time or polynomial-time delay between outputing two solutions [23, 48, 51, 89, 90, 159, 215]. We do not consider enumeration problems in this dissertation, but study a related fundamental combinatorial problem: given a query, can we estimate what the maximum number of solutions can be? It seems that this very basic question was overlooked in the database theory literature, even though estimating the size of the result is important in query optimization [54, 119, 133, 147, 179, 204]. In Chapter 4, we give tight bounds on the maximum number of solutions: the bound is intimately related to the fractional edge cover number of the hypergraph of the query. Although this bound follows relatively easily from known combinatorial results, it was apparently new to the database theory community and inspired a substantial amount of follow up work in the premier venues of database theory [114, 143, 157, 160, 161, 195, 196, 229].

1.6 Publications related to the dissertation

This dissertation is based on six journal publications; it contains almost all results from five of them [126, 176, 184, 185, 188] and the first half of one of them [21]. Three out of these pulications are single author [184, 185, 188].

The results presented in Chapter 2 appeared in *ACM Transactions on Algorithms* [176] (an extended abstract appeared in the proceedings of the SODA 2011 conference [174]). It is joint work with Daniel Lokshtanov and Saket Saurabh; all three authors contributed equally to the publications.

Chapter 3 is based on a single-author publication that appeared in *Theory of Computing* [185] (an extended abstract appeared in the proceedings of the FOCS 2007 conference [180]).

Chapter 4 is based on two publications. Section 4.2 of this chapter is based on the first half of an article that appeared in *ACM Transactions on Algorithms* [126] (an extended abstract appeared in the proceedings of the SODA 2006 conference [124]). It is joint work with Martin Grohe; both authors contributed equally to the publications. Sections 4.3–4.5 of this chapter are based on the first half of an article that appeared in *SIAM Journal on Computing* [21] (an extended abstract appeared in the proceedings of the FOCS 2008 conference [20]). It is joint work with Albert Atserias and Martin Grohe, all three authors contributed equally to the publications.

Chapter 5 is again based on two publications. Section 5.1 is based on the second half of the above-mentioned publications appearing in *ACM Transactions on Algorithms* [126] and SODA 2006 conference [124]) coauthored with Martin Grohe. Section 5.2 is based on a single-author publication that appeared in *ACM Transactions on Algorithms* [184] (an extended abstract appeared in the proceedings of the SODA 2009 conference [183]).

Chapter 6 is based on a single-author publication that appeared in *Journal of the ACM* [188] (an extended abstract appeared in the proceedings of the STOC 2010 conference [186]).

CHAPTER 2

Known algorithms on graphs of bounded treewidth are probably optimal

It is well-known that many NP-hard graph problems can be solved efficiently if the *treewidth* ($\mathrm{tw}(G)$) of the input graph G is bounded. For an example, an expository algorithm to solve Vertex Cover and Independent Set running in time $4^{\mathrm{tw}(G)} \cdot n^{O(1)}$ is described in the algorithms textbook by Kleinberg and Tardos [164], while the book of Niedermeier [197] on fixed-parameter algorithms presents an algorithm with running time $2^{\mathrm{tw}(G)} \cdot n^{O(1)}$. Similar algorithms, with running times on the form $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$ for a constant c, are known for many other graph problems such as Dominating Set, q-Coloring and Odd Cycle Transversal [11, 74, 99, 102, 227]. Algorithms for graph problems on bounded treewidth graphs have found many uses as subroutines in approximation algorithms [24, 84, 93, 171], parameterized algorithms [9, 76, 83, 153, 192, 225], and exact algorithms [103, 194, 216].

In this chapter, we show that any improvement over the currently best known algorithms for a number of well-studied problems on graphs of bounded treewidth would yield a faster algorithm for SAT. In particular, we show the following:

Theorem 2.1. *If there exists an $\epsilon > 0$ such that*

- Independent Set *can be solved in time $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, or*
- Dominating Set *can be solved in time $(3-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, or*
- Max Cut *can be solved in time $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, or*
- Odd Cycle Transversal *can be solved in time $(3-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, or*
- *there is a fixed $q \geq 3$ such that q-*Coloring *can be solved in time $(q-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, or*
- Partition Into Triangles *can be solved in time $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$,*

then n-variable SAT *can be solved in $(2-\delta)^n$ time for some $\delta > 0$.*

Such an algorithm would violate the *Strong Exponential Time Hypothesis* (SETH) of Impagliazzo and Paturi [145] (Hypothesis 1.3 in Chapter 1). Thus, assuming the SETH, the known algorithms for the mentioned problems on graphs of bounded treewidth are essentially the best possible.

Publications. The results presented in this chapter appeared in *ACM Transactions on Algorithms* [176] (an extended abstract appeared in the proceedings of the SODA 2011 conference [174]). It is joint work with Daniel Lokshtanov and Saket Saurabh; all three authors contributed equally to

the publications. The lower bound for INDEPENDENT SET appearing in Section 2.2 was reproduced in the text book of Cygan et. [74] on parameterized algorithms.

Techniques. To show our results we give polynomial time many-one reductions that transform n-variable boolean formulas ϕ to instances of the problems in question. Such reductions are well-known, but for our results we need to carefully control the treewidth of the graphs that our reductions output. A typical reduction creates n gadgets corresponding to the n variables; each gadget has a small constant number of vertices. In most cases, this implies that the treewidth can be bounded by $O(n)$. However, to prove a lower bound of the form $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, we need that the treewidth of the constructed graph is $(1+o(1))n$. Thus we can afford to increase the treewidth by at most one per variable. For lower bounds above $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$, we need even more economical constructions. To understand the difficulty, consider the DOMINATING SET problem, here we want to say that if DOMINATING SET admits an algorithm with running time $(3-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)} = 2^{\log(3-\epsilon)\,\mathrm{tw}(G)} \cdot n^{O(1)}$ for some $\epsilon > 0$, then we can solve SAT on input formulas with n-variables in time $(2-\delta)^n \cdot n^{O(1)}$ for some $\delta > 0$ (when not noted otherwise, logarithms are always base-2 in this disseration). Therefore by naïvely equating the exponent in the previous sentence we get that we need to construct an instance for DOMINATING SET whose treewidth is essentially $\frac{n}{\log 3}$. In other words, each variable should increase treewidth by *less than one*. The main challenge in our reductions is to squeeze out as many combinatorial possibilities per increase of treewidth as possible. In order to control the treewidth of the graphs we construct, we upper bound the *pathwidth* (pw(G)) of the constructed instances and use the fact that for any graph G, $\mathrm{tw}(G) \leq \mathrm{pw}(G)$. Thus all of our lower bounds also hold for problems on graphs of bounded pathwidth.

Related work. In several cases designing the "right algorithm" on graphs of bounded treewidth or pathwidth is not at all obvious. For example: Alber et al. [11] gave a $4^{\mathrm{tw}(G)} \cdot n^{O(1)}$ time algorithm for DOMINATING SET, improving over the natural $9^{\mathrm{tw}(G)} \cdot n^{O(1)}$ algorithm of Telle and Proskurowski [223]. Later, van Rooij et al. [227] observed that one could use fast subset convolution [29] to improve the running time of algorithms on graphs of bounded treewidth. Their results include a $3^{\mathrm{tw}(G)} \cdot n^{O(1)}$ algorithm for DOMINATING SET and a $2^{\mathrm{tw}(G)} \cdot n^{O(1)}$ time algorithm for PARTITION INTO TRIANGLES. Interestingly, the effect of applying subset convolution was that the running time for several graph problems on bounded treewidth graphs became the same as the running time for the problems on graphs of bounded pathwidth. However, the idea of using subset convolution in designing dynamic programming algorithm over graphs of bounded treewidth was not enough to design "optimal algorithms" for several connectivity problems such as HAMILTONIAN PATH and CONNECTED VERTEX COVER. In a seminal paper, Cygan et al. [76] introduced the method of *Cut & Count* and designed the first $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$ time algorithms, where c is a fixed constant, for plethora of connectivity problems including HAMILTONIAN PATH and CONNECTED VERTEX COVER. However, the algorithm for HAMILTONIAN PATH runs in time $4^{\mathrm{tw}(G)} \cdot n^{O(1)}$, which still is the best known algorithm. Later, in a surprising result, Cygan, Kratsch, and Nederlof [75] showed that HAMILTONIAN PATH can be solved in time $(2+\sqrt{2})^{\mathrm{pw}(G)} \cdot n^{O(1)}$ on graphs of bounded pathwidth. The algorithms obtained using *Cut & Count* are randomized. Later, deterministic algorithms with running time $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$, where c is a fixed constant, were designed for connectivity problems [34, 107].

Follow-up work. The problems considered in this article, and the ideas used to resolve them, led to several follow-up publications that showed lower bounds for concrete problems in the parameterized settings [38, 70, 72, 76, 91, 151, 152]. The work of Cygan et al. [76] that introduced the method of *Cut & Count* to design $c^{\mathrm{tw}(G)} \cdot n^{O(1)}$, where c is a fixed constant, for connectivity problems, also showed that the base of exponent in their algorithm are optimal unless the SETH fails. Cygan, Kratsch,

and Nederlof [75] showed that the running time of $(2+\sqrt{2})^{\mathrm{pw}(G)} \cdot n^{O(1)}$ for Hamiltonian Path on graphs of bounded pathwidth is in fact optimal under the SETH. Several other lower bounds for concrete problems were also obtained in [72]. Ideas from the current paper were recently used to design tight lower bounds for r-Dominating Set and Connected Dominating Set on graphs of bounded treewidth [38]. Curticapean and Marx obtained tight lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus under the SETH [70].

Recently, Jaffke and Jansen [152] strengthened our lower bounds for q-Coloring. In particular, they showed that q-Coloring parameterized by the modulator to linear forests (a forest where every connected component is a path), say $\mathrm{lfvs}(G)$, can not be solved in time $(q-\epsilon)^{\mathrm{lfvs}(G)}|V(G)|^{\mathcal{O}(1)}$.

2.1 Preliminaries

In this section we give various definitions which we make use of in the chapter. For basic definitions related to graphs, treewidth, and pathwidth, see Section 1.3. For the purposes of this section, we need an equivalent definition of pathwidth via *mixed search* games. In a mixed search game, a graph G is considered as a system of tunnels. Initially, all edges are contaminated by a gas. An edge is *cleared* by placing searchers at both its end-points simultaneously or by sliding a searcher along the edge. A cleared edge is re-contaminated if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges. A search is a sequence of operations that can be of the following types: (a) placement of a new searcher on a vertex; (b) removal of a searcher from a vertex; (c) sliding a searcher on a vertex along an incident edge and placing the searcher on the other end. A search strategy is winning if after its termination all edges are cleared. The mixed search number of a graph G, denoted by $\mathrm{ms}(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on G. Takahashi, Ueno, and Kajitani [221] obtained the following relationship between $\mathrm{pw}(G)$ and $\mathrm{ms}(G)$, which we use for bounding the pathwidth of the graphs obtained in reduction.

Proposition 2.2 (Takahashi, Ueno, and Kajitani [221])**.** *For a graph G,* $\mathrm{pw}(G) \leq \mathrm{ms}(G) \leq \mathrm{pw}(G)+1$.

An instance to SAT consists of a boolean formula $\phi = C_1 \wedge \cdots \wedge C_m$ over n variables $\{v_1, \ldots, v_n\}$ where each clause C_i is OR of one or more literals of variables. We also denote a clause C_i by the set $\{\ell_1, \ell_2, \ldots, \ell_c\}$ of its literals and denote by $|C_i|$ the number of literals in C_i. An assignment τ to the variables is an element of $\{0,1\}^n$, and it satisfies the formula ϕ if for every clause C_i there is literal that is assigned 1 by τ. We say that a variable v_i satisfies a clause C_j if there exists a literal corresponding to v_i in $\{\ell_1, \ell_2, \ldots, \ell_c\}$ and it is set to 1 by τ. A group of variables satisfy a clause C_j if there is a variable that satisfies the clause C_j. All the sections in this chapter follow the following pattern: definition of the problem; statement of the lower bound; construction used in the reduction; correctness of the reduction; and the upper bound on the pathwidth of the resultant graph.

2.2 Independent Set

An *independent set* of a graph G is a set $S \subseteq V(G)$ such that $G[S]$ contains no edges. In the Independent Set problem we are given a graph G and the objective is to find an independent set of maximum size.

We first sketch the main idea of the proof. We give the reduction from an arbitrary SAT instance on n variables and m clauses. The idea is to create a family of n very long paths $P_1, P_2, \ldots, P_n$ of even length, corresponding to variables $x_1, x_2, \ldots, x_n$. Assume for now that on each of these

paths the solution is allowed to make one of two choices: the independent set either contains all the odd-indexed vertices, or all the even-indexed vertices. Then for every clause we construct a clause verification gadget and attach it to some place in the family. The gadget is adjacent to paths corresponding to variables appearing in the clause, and the attachment points reflect whether the variable's appearance is positive or negative. The role of the clause gadget is to verify that the clause is satisfied. Satisfaction of the clause corresponds to the condition that at least one of the attachment points of the clause gadget needs to be *not* chosen into the constructed independent set; hence the clause gadget needs to have the following property: the behavior inside the gadget can be set optimally if and only if at least one of the attachment points is free. It is possible to construct a gadget with exactly this property, and moreover the gadget has constant pathwidth, so it does not increase much the width of the whole construction.

One technical problem that we still need to overcome is the first technical assumption about the choices the solution makes on the paths P_i. It is namely not true that on a path of even length there are only two maximum-size independent sets: the odd-indexed vertices and the even-indexed vertices. The solution can first start with picking only odd-indexed vertices, then make a gap of two vertices, and continue further with even-indexed vertices. Thus, on each path there can be one "cheat" where the solution flips from odd indices to even indices. The solution to this problem is a remarkably simple trick that is commonly used in similar reductions. We namely repeat the whole sequence of clause gadgets $n+1$ times, which ensures that at most n copies are spoiled by possible cheats, and hence at least one of the copies is attached to an area where no cheat happens, and hence the behavior of the solution on the paths P_i correctly encodes some satisfying assignment of the variable set. This concludes the sketch and we move towards giving the formal proof.

Theorem 2.3. *If* Independent Set *can be solved in* $(2-\epsilon)^{\mathrm{tw}(G)} \cdot n^{O(1)}$ *for some* $\epsilon > 0$ *then* SAT *can be solved in* $(2-\delta)^n \cdot n^{O(1)}$ *time for some* $\delta > 0$.

Construction. Given an instance ϕ of SAT, we construct a graph G as follows (see Figure 2.1). We assume that every clause has an even number of variables: if not, we can add a single variable to all odd size clauses and force this variable to false. First we describe the construction of clause gadgets. For a clause $C = \{\ell_1, \ell_2, \ldots, \ell_c\}$, we introduce a gadget $\widehat{C}$ as follows. We take two paths, $CP = cp_1, cp_2 \ldots, cp_c$ and $CP' = cp'_1, cp'_2 \ldots cp'_c$ having c vertices each, and connect cp_i with cp'_i for every i. For each literal ℓ_i, we introduce a vertex ℓ_i in $\widehat{C}$ and make it adjacent to cp_i and cp'_i. Finally we add two vertices c_{start} and c_{end}, such that c_{start} is adjacent to cp_1 and c_{end} is adjacent to cp_c. Observe that the size of the maximum independent set of $\widehat{C}$ is $c+2$. Also, since c is even, any independent set of size $c+2$ in $\widehat{C}$ must contain at least one vertex in $C = \{\ell_1, \ell_2, \ldots, \ell_c\}$. Finally, notice that for any i, there is an independent set of size $c+2$ in $\widehat{C}$ that contains ℓ_i and none of ℓ_j for $j \neq i$.

We first construct a graph G_1. We introduce n paths $P_1, \ldots, P_n$, each path has $2m$ vertices. Let the vertices of the path P_i be $p_i^1 \ldots p_i^{2m}$. The path P_i corresponds to the variable v_i. For every clause C_i of ϕ, we introduce a gadget $\widehat{C}_i$. Now, for every variable v_i, if v_i occurs positively in C_j, we add an edge between p_i^{2j} and the literal corresponding to v_i in $\widehat{C}_j$. If v_i occurs negatively in C_j, we add an edge between p_i^{2j-1} and the literal corresponding to v_i in $\widehat{C}_j$. Now we construct the graph G as follows. We take $n+1$ copies of G_1, call them G_1, ..., G_{n+1}. For every $i \leq n$, we connect G_i and G_{i+1} by connecting p_j^{2m} in G_i with p_j^1 in G_{i+1} for every $j \leq n$. This way, the paths P_j in each of the n copies G_i together form a long path of $2m(n+1)$ vertices. This concludes the construction of G.

Lemma 2.4. *If* ϕ *is satisfiable, then* G *has an independent set of size* $(mn + \sum_{i \leq m}(|C_i|+2))(n+1)$.

Proof. Consider a satisfying assignment to ϕ. We construct an independent set I in G. For every variable v_i, if v_i is set to true, then pick all the vertices on odd positions from all copies of P_i, that

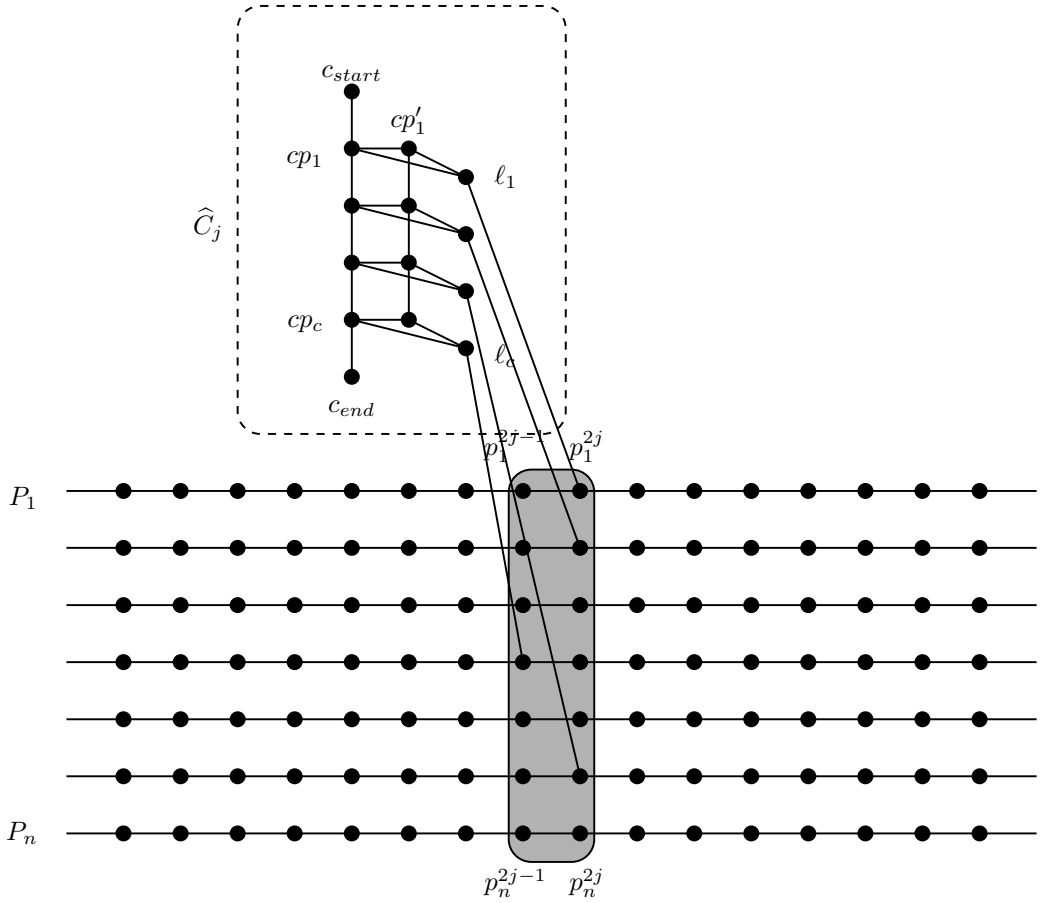

Figure 2.1: Reduction to INDEPENDENT SET: clause gadget $\widehat{C}_j$ attached to the n paths representing the variables.

is p_i^1, p_i^3, p_i^5 and so on. If v_i is false then pick all the vertices on even positions from all copies of P_i, that is p_i^2, p_i^4, p_i^6 and so on. It is easy to see that this is an independent set of size $mn(n+1)$ containing vertices from all the paths. We will now consider the gadget $\widehat{C}_j$ corresponding to a clause C_j. We will only consider the copy of $\widehat{C}_j$ in G_1 as the other copies can be dealt identically. Let us choose a true literal ℓ_a in C_j and let v_i be the corresponding variable. Consider the vertex ℓ_a in $\widehat{C}_j$. If v_i occurs positively in C_j, then v_i is true. Then I does not contain p_i^{2j}, the only neighbour of ℓ_a outside of $\widehat{C}_j$. On the other hand if v_i occurs negatively in C_j, then v_i is false. In this case I does not contain p_i^{2j-1}, the only neighbour of ℓ_a outside of $\widehat{C}_j$. There is an independent set of size $|C_j|+2$ in $\widehat{C}$ that contains ℓ_a and none out of ℓ_b for any $b \neq a$. We add this independent set to I and proceed in this manner for every clause gadget. By the end of the process $(\sum_{i\leq m}(|C_i|+2))(n+1)$ vertices from clause gadgets are added to I, yielding that the size of I is $(mn + \sum_{i\leq m}(|C_i|+2))(n+1)$, concluding the proof. □

Lemma 2.5. *If G has an independent set of size $(mn+\sum_{i\leq m}(|C_i|+2))(n+1)$, then ϕ is satisfiable.*

Proof. Consider an independent set of G of size $(mn + \sum_{i\leq m}|C_i|+2)(n+1)$. Set I can contain at most m vertices from each copy of P_i for every $i \leq n$ and at most $|C_j|+2$ vertices from each copy of the gadget C_j. Since I must contain at least that many vertices from each path and clause gadget

in order to contain at least $(mn + \sum_{i \leq m} |C_i| + 2)(n+1)$ vertices, it follows that I has exactly m vertices in each copy of each path P_i and exactly $|C_j| + 2$ vertices in each copy of each clause gadget $\widehat{C}_j$. For a fixed j, consider the $n+1$ copies of the path P_j. Since P_j in G_i is attached to P_j in G_{i+1}, these $n+1$ copies of P_i together form a path P having $2m(n+1)$ vertices. Since $|I \cap P| = m(n+1)$ it follows that $I \cap P$ must contain every second vertex of P, except possibly in one position where $I \cap P$ skips two vertices of P. There are only n paths and $n+1$ copies of G_1, hence the pigeon-hole principle implies that in some copy G_y of G_1, I contains every second vertex on every path P_i. From now onwards we only consider such a copy G_y.

In G_y, for every $i \leq n$, I contains every second vertex of P_i. We make an assignment to the variables of ϕ as follows. If I contains all the odd numbered vertices of P_i then v_i is set to true, otherwise I contains all the even numbered vertices of P_i and v_i is set to false. We argue that this assignment satisfies ϕ. Indeed, consider any clause C_j, and look at the gadget $\widehat{C}_j$. We know that I contains $|C_j| + 2$ vertices from $\widehat{C}_j$ and hence I must contain a vertex ℓ_a in $\widehat{C}_j$ corresponding to a literal of C_j. Suppose ℓ_a is a literal of v_i. Since I contains ℓ_a, if ℓ_a occurs positively in C_j, then I can not contain p_i^{2j} and hence v_i is true. Similarly, if ℓ_a occurs negatively in C_j then I can not contain p_i^{2j-1} and hence v_i is false. In both cases v_i satisfies C_j and hence all clauses of ϕ are satisfied by the assignment. □

Lemma 2.6. $\mathrm{pw}(G) \leq n + 4$.

Proof. We give a mixed search strategy to clean G using $n+3$ searchers. For every i we place a searcher on the first vertex of P_i in G_1. The n searchers slide along the paths $P_1, \ldots P_n$ in m rounds. In round j each searcher i starts on p_i^{2j-1}. Then, for every variable v_i that occurs positively in C_j, the searcher i slides forward to p_i^{2j}. Observe that at this point there is a searcher on every neighbour of the gadget $\widehat{C}_j$. This gadget can now be cleaned with 3 additional searchers. After $\widehat{C}_j$ is clean, the additional 3 searchers are removed, and each of the n searchers on the paths $P_1, \ldots P_n$ slides forward along these paths, such that searcher i stands on $p_i^{2(j+1)}$. At that point, the next round commences. When the searchers have cleaned G_1 they slide onto the first vertex of $P_1 \ldots P_n$ in G_2. Then they proceed to clean $G_2, \ldots, G_{n+1}$ in the same way that G_1 was cleaned. Now applying Proposition 2.2 we get that $\mathrm{pw}(G) \leq n + 4$. □

The construction, together with Lemmata 2.4, 2.5 and 2.6 proves Theorem 2.3.

2.3 Dominating Set

A *dominating set* of a graph G is a set $S \subseteq V(G)$ such that $V(G) = N[S]$. In the Dominating Set problem we are given a graph G and the objective is to find a dominating set of minimum size.

The basic idea for this reduction is similar to the one for Independent Set. However, we need one more new idea here, which will also be used in other reductions. We group variables into an appropriate number of groups of size at most $\beta = \lfloor \log 3^p \rfloor$, where p is a constant depending only on ϵ. Then, for every group we make a gadget such that an assignment on the group should correspond to a selection on the gadget. These group gadgets are then connected to clause gadgets so that every assignment on the group that satisfies the clause results in some desired outcome.

Theorem 2.7. *If* Dominating Set *can be solved in* $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ *time for some* $\epsilon > 0$ *then* SAT *can be solved in* $(2-\delta)^n \cdot n^{O(1)}$ *time for some* $\delta > 0$.

Construction. Given $\epsilon < 1$ and an instance ϕ to SAT we construct a graph G as follows. We first choose an integer p depending only on ϵ. Exactly how p is chosen will be discussed in the

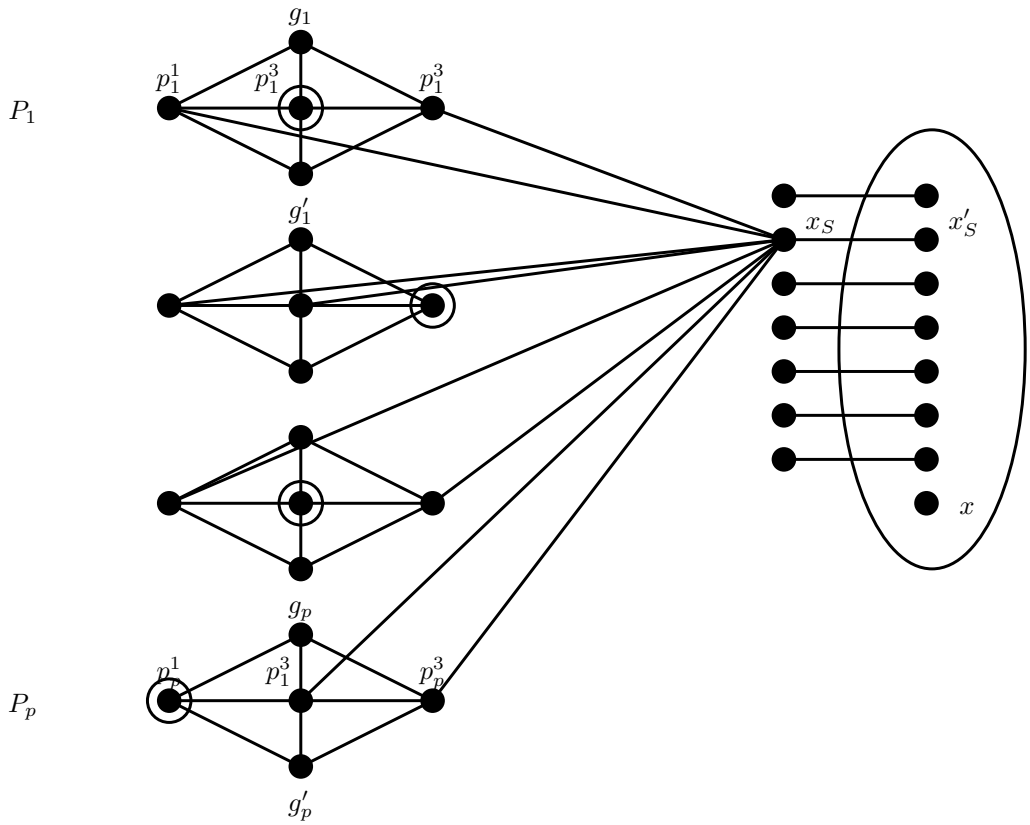

Figure 2.2: Reduction to DOMINATING SET: group gadget $\widehat{B}$. The set S is shown by the circled vertices.

proof of Theorem 2.7. We group the variables of ϕ into groups $F_1, F_2, \ldots, F_t$, each of size at most $\beta = \lfloor \log 3^p \rfloor$. Hence $t = \lceil n/\beta \rceil$. We now proceed to describe a "group gadget" $\widehat{B}$, which is central in our construction.

To build the group gadget $\widehat{B}$, we introduce p paths $P_1, \ldots, P_p$, where the path P_i contains the vertices p_i^1, p_i^2 and p_i^3 (see Figure 2.2). To each path P_i we attach two *guards* g_i and g_i', both of which are neighbours to p_i^1, p_i^2 and p_i^3. When the gadgets are attached to each other, the guards will not have any neighbours outside of their own gadget $\widehat{B}$, and will ensure that at least one vertex out of p_i^1, p_i^2 and p_i^3 are chosen in any minimum size dominating set of G. Let P be the vertex set containing all the vertices on the paths $P_1, \ldots, P_p$. For every subset S of P that picks *exactly one* vertex from each path P_i, we introduce two vertices x_S and x_S', where x_S is adjacent to all vertices of $P \setminus S$ (all those vertices that are on paths and not in S) and x_S' is only adjacent to x_S. We conclude the construction of $\widehat{B}$ by making all the vertices x_S' (for every set S) adjacent to each other, that is making them into a clique, and adding a guard x adjacent to x_S' for every set S. In other words, the x_S''s together with x form a clique and all the neighbors of x reside in this clique.

We construct the graph G as follows (see Figure 2.3). For every group F_i of variables, we introduce $m(2pt+1)$ copies of the gadget $\widehat{B}$, call them $\widehat{B}_i^j$ for $1 \leq j \leq m(2pt+1)$. We can imagine these $t \cdot m(2pt+1)$ gadgets arranged in t rows and $m(2pt+1)$ columns, with the columns being divided into $2pt+1$ *regions* of m columns each. For every fixed $i \leq t$, we connect the gadgets $\widehat{B}_i^1, \widehat{B}_i^2 \ldots, \widehat{B}_i^{m(2pt+1)}$ in a path-like manner. In particular, for every $j < m(2pt+1)$ and every $\ell \leq p$ we make an edge between p_ℓ^3 in the gadget $\widehat{B}_i^j$ with p_ℓ^1 in the gadget $\widehat{B}_i^{j+1}$. Now we introduce two

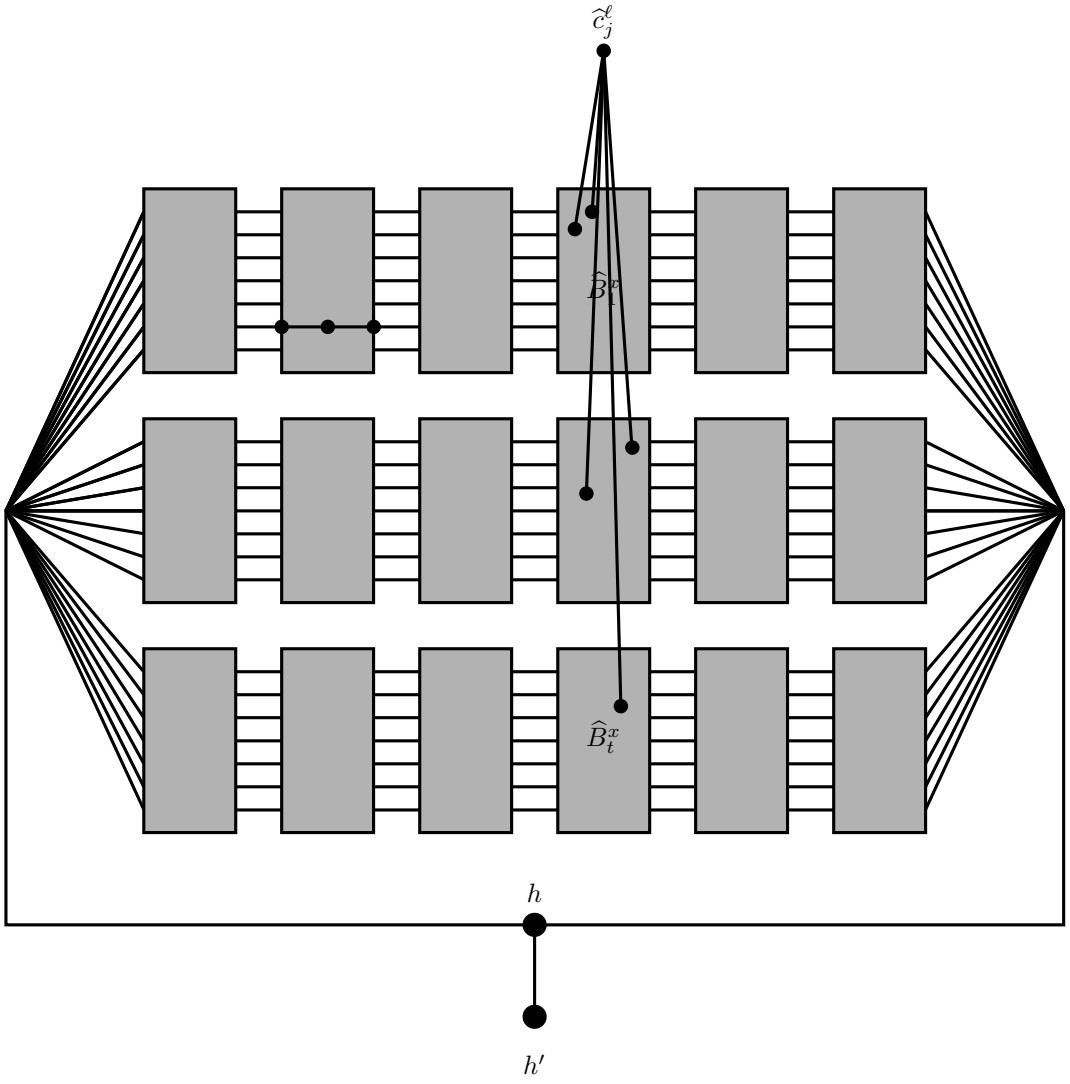

Figure 2.3: Reduction to Dominating Set: arranging the group gadgets. Note that $x = m\ell + j$, thus $\widehat{c}_j^{\ell}$ is attached to vertices in $\widehat{B}_1^x, \ldots, \widehat{B}_t^x$.

new vertices h and h', with h adjacent to h', p_j^1 in $\widehat{B}_i^1$ for every $i \leq t$, $j \leq p$ and to p_j^3 in $\widehat{B}_i^{m(2pt+1)}$ for every $i \leq t$, $j \leq p$. That is, for all $1 \leq i \leq t$, h is adjacent to the first and last vertices of "long paths" obtained after connecting the gadgets $\widehat{B}_i^1, \widehat{B}_i^2 \ldots, \widehat{B}_i^{m(2pt+1)}$ in a path-like manner.

For every $1 \leq i \leq t$ and to every assignment of the variables in the group F_i, we designate a subset S of P in the gadget $\widehat{B}$ that picks exactly one vertex from each path P_j. Since there are at most 2^β different assignments to the variables in F_i, and there are $3^p \geq 2^\beta$ such sets S, we can assign a *unique* set to each assignment. Of course, the same set S can correspond to one assignment of the group F_1 and some other assignment of the group F_2. Recall that the clauses of ϕ are $C_1, \ldots, C_m$. For every clause C_j we introduce $2pt + 1$ vertices $\widehat{c}_j^{\ell}$, one for each $0 \leq \ell < 2pt + 1$, corresponding to the $2pt + 1$ regions. The vertex $\widehat{c}_j^{\ell}$ will be connected to the gadgets $\widehat{B}_i^{m\ell+j}$ for every $1 \leq i \leq t$ (which appear in the ℓ-th region). In particular, for every assignment of the variables in the group F_i that satisfy the clause C_j, we consider the subset S of P that corresponds to the assignment. For every $0 \leq \ell < 2pt + 1$, we make x'_S in $\widehat{B}_i^{m\ell+j}$ adjacent to $\widehat{c}_j^{\ell}$. The best way to view this is that every

clause C_j has $2pt+1$ private gadgets in the i-row, $\widehat{B}_i^j, \widehat{B}_i^{m+j}, \ldots, \widehat{B}_i^{m2pt+j}$, one in each region. Now we have $2pt+1$ vertices corresponding to the clause C_j, each connected to one of these gadgets. This concludes the construction of G.

Lemma 2.8. *If ϕ has a satisfying assignment, then G has a dominating set of size $(p+1)tm(2pt+1)+1$.*

Proof. Given a satisfying assignment to ϕ, we construct a dominating set D of G that contains the vertex h and *exactly* $p+1$ vertices in each gadget $\widehat{B}_i^j$. For each group F_i of variables we consider the set S that corresponds to the restriction of the assignment to the variables in F_i. From each gadget $\widehat{B}_i^j$ we add the set S to D and also the vertex x'_S to D. It remains to argue that D is indeed a dominating set. Clearly the size is bounded by $(p+1)tm(2pt+1)+1$, as the number of gadgets is $tm(2pt+1)$.

For a fixed $i \leq t$ and j consider the vertices on the path P_j in the gadgets $\widehat{B}_i^\ell$ for every $\ell \leq m(2pt+1)$. Together these vertices form a path of length $3m(2pt+1)$ and every third vertex of this path is in S. Thus, all vertices on this path are dominated by other vertices on the path, except perhaps for the first and last one. Both these vertices, however, are dominated by h.

Now, fix some $i \leq t$ and $\ell \leq m(2pt+1)$ and consider the gadget $\widehat{B}_i^\ell$. Since D contains some vertex on the path P_j, we have that for every j both g_j and g'_j are dominated. Furthermore, for every set S^* not equal to S that picks exactly one vertex from each P_j, vertex x_{S^*} is dominated by some vertex on some P_j—namely by all vertices in $S \setminus S^* \neq \emptyset$. The last assertion follows since x_{S^*} is connected to all the vertices on the paths except S^*. On the other hand, x_S is dominated by x'_S, and x'_S also dominates all the other vertices x'_{S^*} for $S^* \neq S$, as well as the guard x.

The only vertices not yet accounted for are the vertices $\widehat{c}_j^\ell$ for every $j \leq m$ and $\ell < 2pt+1$. Fix a j and a ℓ and consider the clause C_j. This clause contains a literal set to true, and this literal corresponds to a variable in the group F_i for some $i \leq t$. Of course, the assignment to F_i satisfies C_j. Let S be the set corresponding to this assignment of F_i. By the construction of D, the dominating set contains x'_S in $\widehat{B}_i^{m\ell+j}$ and x'_S is adjacent to $\widehat{c}_j^\ell$. This concludes the proof. □

Lemma 2.9. *If G has a dominating set of size $(p+1)tm(2pt+1)+1$, then ϕ has a satisfying assignment.*

Proof. Let D be a dominating set of G of size at most $(p+1)tm(2pt+1)+1$. Since D must dominate h', without loss of generality we can assume that D contains h. Furthermore, inside every gadget $\widehat{B}_i^\ell$, D must dominate all the guards, namely g_j and g'_j for every $j \leq p$, and also x. Thus D contains at least $p+1$ vertices from each gadget $\widehat{B}_i^\ell$ which in turn implies that D contains exactly $p+1$ vertices from each gadget $\widehat{B}_i^\ell$. The only way D can dominate g_j and g'_j for every j and in addition dominate x with only $p+1$ vertices if D has one vertex from each P_j, $j \leq p$ and in addition contains some vertex in $N[x]$. Let S be $D \cap P$ in $\widehat{B}_i^\ell$. Observe that x_S is not dominated by $D \cap S$. The only vertex in $N[x]$ that dominates x_S is x'_S and hence D contains x'_S.

Now we want to show that for every $1 \leq i \leq t$ there exists *one* $0 \leq \ell \leq 2tp$ such that for fixed i, $D \cap P$ is same in all the gadgets $\widehat{B}_i^{m\ell+r}$ for every $1 \leq r \leq m$, i.e., it is the same in every gadget of the i-th row in the ℓ-th region. Consider a gadget $\widehat{B}_i^\ell$ and its follower, $\widehat{B}_i^{\ell+1}$. Let S be $D \cap P$ in $\widehat{B}_i^\ell$ and S' be $D \cap P$ in $\widehat{B}_i^{\ell+1}$. Observe that if S contains p_j^a in $\widehat{B}_i^\ell$ and p_j^b in $\widehat{B}_i^{\ell+1}$ then we must have $b \leq a$. We call a consecutive pair *bad* if for some $j \leq p$, D contains p_j^a in $\widehat{B}_i^\ell$ and p_j^b in $\widehat{B}_i^{\ell+1}$ and $b < a$. Hence for a fixed i, we can at most have $2p$ consecutive bad pairs, spoiling at most $2p$ regions. Now we mark all the bad pairs that occur among the gadgets corresponding to some F_i. This way we can mark only $2tp$ bad pairs. Thus, by the pigeon hole principle, there exists an $\ell \in \{0, \ldots, 2tp\}$ such that there are no bad pairs in $\widehat{B}_i^{m\ell+r}$ for all $1 \leq i \leq t$ and $1 \leq r \leq m$.

We make an assignment ϕ by reading off $D \cap P$ in each gadget $\widehat{B}_i^{m\ell+1}$. In particular, for every group F_i, we consider $S = D \cap P$ in the gadget $\widehat{B}_i^{m\ell+1}$. This set S corresponds to an assignment of F_i, and this is the assignment of F_i that we use. It remains to argue that every clause C_r is satisfied by this assignment.

Consider the vertex $\widehat{c}_\ell^r$. We know that it is dominated by some x'_S in a gadget $\widehat{B}_i^{m\ell+r}$. The set S corresponds to an assignment of F_i that satisfies the clause C_r. Because $D \cap P$ remains unchanged in all gadgets from $\widehat{B}_i^{m\ell+1}$ to $\widehat{B}_i^{m\ell+r}$, this is exactly the assignment ϕ restricted to the group F_i. This concludes the proof. □

Lemma 2.10. $\mathrm{pw}(G) \leq tp + \mathcal{O}(3^p)$

Proof. We give a mixed search strategy to clean the graph with $tp + \mathcal{O}(3^p)$ searchers. For a gadget $\widehat{B}$ we call the vertices p_j^1 and p_j^3, $1 \leq j \leq p$, as *entry vertices* and *exit vertices* respectively. We search the graph in $m(2tp+1)$ rounds. In the beginning of round ℓ there are searchers on the entry vertices of the gadgets $\widehat{B}_i^\ell$ for every $i \leq t$. Let $1 \leq a \leq m$ and $0 \leq b < 2tp+1$ be integers such that $\ell = a + mb$. We place a searcher on $\widehat{c}_a^b$. Then, for each i between 1 and p in turn we first put searchers on all vertices of $\widehat{B}_i^\ell$ and then remove all the searchers from $\widehat{B}_i^\ell$ except for the ones standing on the exit vertices. After all gadgets $\widehat{B}_1^\ell \dots \widehat{B}_t^\ell$ have been cleaned in this manner, we can remove the searcher from $\widehat{c}_a^b$. To commence the next round, the searchers slide from the exit positions of $\widehat{B}_i^\ell$ to the entry positions of $\widehat{B}_i^{\ell+1}$ for every i. In total, at most $tp + |V(\widehat{B})| + 1 \leq tp + \mathcal{O}(3^p)$ searchers are used simultaneously. This together with Proposition 2.2 give the desired upperbound on the pathwidth. □

Proof (of Theorem 2.7). Suppose Dominating Set can be solved in $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)} = 3^{\lambda \mathrm{pw}(G)} \cdot n^{O(1)}$ time, where $\lambda = \log_3(3-\epsilon) < 1$. We choose p large enough such that $\lambda \cdot \frac{p}{\lfloor p \log 3 \rfloor} = \frac{\delta'}{\log 3}$ for some $\delta' < 1$. Given an instance of SAT, we construct an instance of Dominating Set using the above construction and the chosen value of p. Then we solve the Dominating Set instance using the $3^{\lambda \mathrm{pw}(G)} \cdot n^{O(1)}$ time algorithm. Correctness is ensured by Lemmata 2.8 and 2.9. Lemma 2.10 yields that the total time taken is upper bounded by $3^{\lambda \mathrm{pw}(G)} \cdot n^{O(1)} \leq 3^{\lambda(tp+f(\lambda))} \cdot n^{O(1)} \leq 3^{\lambda \frac{np}{\lfloor p \log 3 \rfloor}}) \cdot n^{O(1)} \leq 3^{\delta' \frac{n}{\log 3}} \cdot n^{O(1)} \leq 2^{\delta'' n} \cdot n^{O(1)} = (2-\delta)^n \cdot n^{O(1)}$, for some $\delta'', \delta < 1$. This concludes the proof. □

2.4 Max Cut

A *cut* in a graph G is a partition of $V(G)$ into V_0 and V_1. The *cut-set* of the cut is the set of edges whose one end point is in V_0 and the other in V_1. We say that an edge is *crossing* this cut if it has one endpoint in V_0 and one in V_1, that is, the edge is in the cut-set. The *size* of the cut is the number of edges in G which are crossing this cut. If the edges of G have positive integer weights, then the *weight* of the cut is the sum of the weights of edges that are crossing the cut. In the Max Cut problem, we are given a graph G together with an integer t and asked whether there is a cut of G of size at least t. In the Weighted Max Cut problem every edge has a positive integer weight and the objective is to find a cut of weight at least t.

Theorem 2.11. *If* Max Cut *can be solved in* $(2-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ *for some* $\epsilon > 0$*, then* SAT *can be solved in* $(2-\delta)^n \cdot n^{O(1)}$ *time for some* $\delta > 0$*.*

Construction. Given an instance ϕ of SAT, we first construct an instance G_w of Weighted Max Cut as follows. We later explain how to obtain an instance of unweighted Max Cut from here.

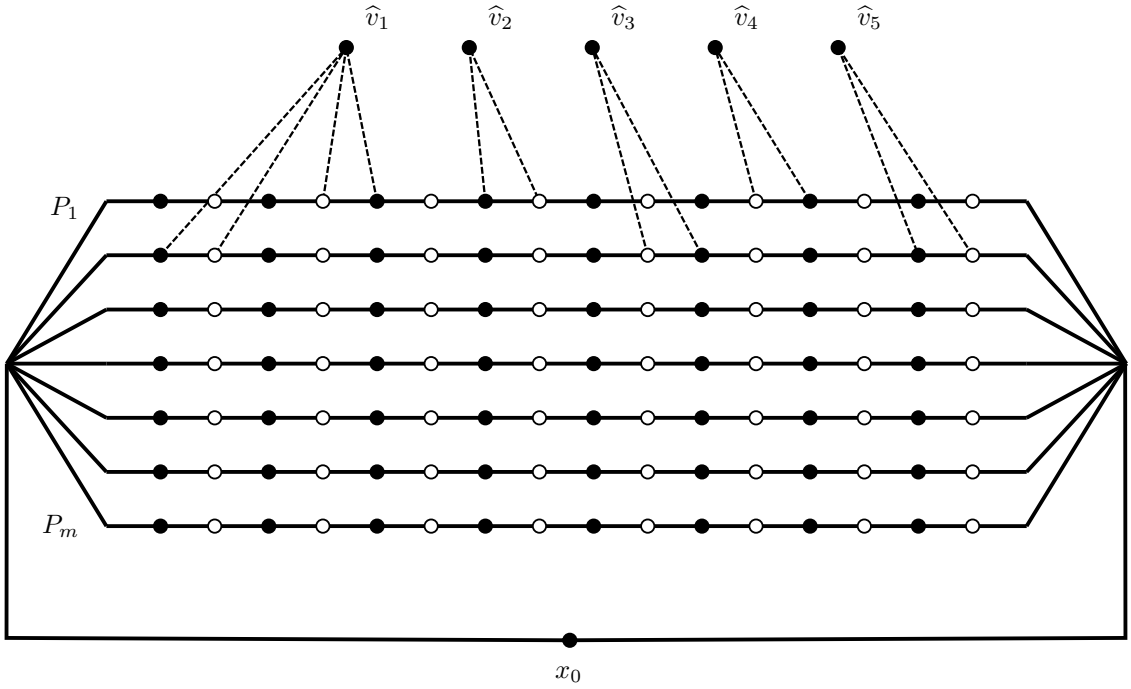

Figure 2.4: Reduction to MAX CUT. The dashed edges have weight 1 and all the other edges have weight $3n$. The odd and even positions of the paths P_j are shown by black and white, respectively. As an example, we show potential connections corresponding to the clauses $C_1 = (v_1 \vee \bar{v}_2 \vee v_4)$ and $C_2 = (\bar{v}_1 \vee v_3 \vee \bar{v}_5)$.

We start with introducing a vertex x_0. Without loss of generality, we will assume that $x_0 \in V_0$ in every solution. We introduce a vertex $\widehat{v}_i$ for each variable v_i. For every clause C_j, we create a gadget as follows. We introduce a path $\widehat{P}_j$ having $4|C_j|$ vertices. All the edges on $\widehat{P}_j$ have weight $3n$. Now, we make the first and last vertex of $\widehat{P}_j$ adjacent to x_0 with an edge of weight $3n$. Thus the path $\widehat{P}_j$ plus the edges from the first and last vertex of $\widehat{P}_j$ to x_0 form an odd cycle $\widehat{C}_j$. We will say that the first, third, fifth, etc, vertices are on *odd positions* on $\widehat{P}_j$ while the remaining vertices are on *even positions*. For every variable v_i that appears positively in C_j, we select a vertex p at an even position (but not the last vertex) on $\widehat{P}_j$ and make $\widehat{v}_i$ adjacent to p and p's successor on $\widehat{P}_j$ with edges of weight 1. For every variable v_i that appears negatively in C_j we select a vertex p at an odd position on $\widehat{P}_j$ and make $\widehat{v}_i$ adjacent to p and p's successor on $\widehat{P}_j$ with edges of weight 1. We make sure that each vertex on $\widehat{P}_j$ receives an edge at most once in this process. There are more than enough vertices on $\widehat{P}_j$ to accommodate all the edges incident to vertices corresponding to variables in the clause C_j. We create such a gadget for each clause and set $t = m + (12n+1)\sum_{j=1}^{m} |C_j|$. This concludes the construction.

Lemma 2.12. *If ϕ is satisfiable, then G_w has a cut of weight at least t.*

Proof. Suppose ϕ is satisfiable. We put x_0 in V_0 and for every variable v_i we put $\widehat{v}_i$ in V_1 if v_i is true and $\widehat{v}_i$ in V_0 if v_i is false. For every clause C_j we proceed as follows. Let us choose a true literal of C_j and suppose that this literal corresponds to a vertex p_j on $\widehat{P}_j$. We put the first vertex on $\widehat{P}_j$ in V_1, the second in V_0 and then we proceed along $\widehat{P}_j$ putting every second vertex into V_1 and V_0 until we reach p_j. The successor p'_j of p_j on $\widehat{P}_j$ is put into the same set as p_j. Then we continue

along $\widehat{P_j}$ putting every second vertex in V_1 and V_0. Notice that even though C_j may contain more than one literal that is set to true, we only select one vertex p_j from the path $\widehat{P}_j$ and put p_j and its successor on the same side of the partition. It remains to argue that this cut has weight at least t.

For every clause C_j all edges on the path $\widehat{P}_j$ except for $p_jp'_j$ are crossing, and the two edges to x_0 from the first and last vertex of $\widehat{P}_j$ are crossing as well. These edges contribute $12n|C_j|$ to the weight of the cut. We know that p_j corresponds to a literal that is set to true, and this literal corresponds to a variable v_i. If v_i occurs positively in C_j, then $\widehat{v}_i \in V_1$ and p_j is on an even position of $\widehat{P}_j$. Thus both p_j and its successor p'_j are in V_0 and hence both $\widehat{v}_ip_j$ and $\widehat{v}_ip'_j$ are crossing, contributing 2 to the weight of the cut. For each of the remaining variables $v_{i'}$ appearing in C_j, one of the two neighbours of $\widehat{v}_{i'}$ on $\widehat{P}_j$ appear in V_0 and one in V_1, so exactly one edge from $v_{i'}$ to $\widehat{P}_j$ is crossing. Thus the total weight of the cut is $t = \sum_{j=1}^{m}(12n|C_j| + |C_j| + 1) = m + (12n+1)\sum_{j=1}^{m}|C_j|$. This completes the proof. □

Lemma 2.13. *If G_w has a cut of weight at least t, then ϕ is satisfiable.*

Proof. Let (V_0, V_1) be a cut of G of maximum weight, hence the weight of this cut is at least t. Without loss of generality, let $x_0 \in V_0$. For every clause C_j, at least one edge of the odd cycle $\widehat{C}_j$ is not crossing. If more than one edge of this cycle is not crossing, then the total weight of the cut edges incident to the path $\widehat{P}_j$ is at most $3n(4|C_j| - 1) + 2n < 12n|C_j|$. In this case, we could change the partition (V_0, V_1) such that all edges of $\widehat{P}_j$ are crossing and the first vertex of $\widehat{P}_j$ is in V_1. Using the new partition the weight of the crossing edges in the cycle $\widehat{C}_j$ is at least $12n|C_j|$ and the edges not incident to $\widehat{P}_j$ are unaffected by the changes. This contradicts that (V_0, V_1) was a maximum weight cut. Thus it follows that exactly one edge of $\widehat{C}_j$ is not crossing.

Given the cut (V_0, V_1), we set each variable v_i to true if $\widehat{v}_i \in V_1$ and v_i to false otherwise. Consider a clause C_j and a variable v_i that appears in C_j. Let uv be the edge of $\widehat{C}_j$ that is not crossing. If there is a vertex $\widehat{v}_i$ adjacent to both u and v, then it is possible that both $\widehat{v}_iu$ and $\widehat{v}_iv$ are crossing. For every other variable $v_{i'}$ in C_j, at most one of the edges from $\widehat{v}_{i'}$ to $\widehat{P}_j$ is crossing. Thus, the weight of the edges that are crossing in the gadget $\widehat{C}_j$ is at most $(12n+1)|C_j| + 1$. Hence, to find a cut-set of weight at least t in G, we need to have crossing edges in $\widehat{C}_j$ with sum of their weights exactly equal to $12n|C_j| + |C_j| + 1$. It follows that there is a vertex $\widehat{v}_i$ adjacent to both u and v such that both $\widehat{v}_iu$ and $\widehat{v}_iv$ are crossing.

If v_i occurs in C_j positively, then u is on an even position and hence, $u \in V_0$. Since $\widehat{v}_iu$ is crossing it follows that v_i is true and C_j is satisfied. On the other hand, if v_i occurs in C_j negated then u is on an odd position and hence, $u \in V_1$. Since $\widehat{v}_iu$ is crossing it follows that v_i is false and C_j is satisfied. As this holds for each clause individually, this concludes the proof. □

For every edge $e \in E(G_w)$, let w_e be the weight of e in G_w. We construct an unweighted graph G from G_w by replacing every edge $e = uv$ by w_e paths from u to v on three edges. Let W be the sum of the edge weights of all edges in G_w.

Lemma 2.14. *G has a cut of size $2W + t$ if and only if G_w has a cut of weight at least t.*

Proof. Given a partition of $V(G_w)$, we partition $V(G)$ as follows. The vertices of G that also are vertices of $V(G)$ are partitioned in the same way as in $V(G_w)$. On each path of length 3, if the endpoints of the path are in different sets we can partition the middle vertices of the path such that all edges are cut. If the endpoints are in the same set we can only partition the middle vertices such that 2 out of the 3 edges are cut. The reverse direction is similar. □

Lemma 2.15. $\mathrm{pw}(G) \leq n + 5$.

Proof. We give a search strategy to clean G with $n+5$ searchers. We place one searcher on each vertex $\widehat{v}_i$ and one searcher on x_0. Then one can search the gadgets $\widehat{C_j}$ one by one. In G_w it is sufficient to use 2 searchers for each $\widehat{C_j}$, whereas in G after the edges have been replaced by multiple paths on three edges, we need 4 searchers. This combined with Proposition 2.2 gives the desired upper bound on the pathwidth of the graph. □

The construction, together with Lemmata 2.12, 2.13, 2.14 and 2.15 proves Theorem 2.11.

2.5 Graph Coloring

A q-coloring of G is a function $\mu : V(G) \to [q]$. A q-coloring μ of G is *proper* if for every edge $uv \in E(G)$ we have $\mu(u) \neq \mu(v)$. In the q-Coloring problem we are given as input a graph G and the objective is to decide whether G has a proper q-coloring. In the List Coloring problem, every vertex v is given a list $L(v) \subseteq [q]$ of admissible colors. A *proper list coloring* of G is a function $\mu : V(G) \to [q]$ such that μ is a proper coloring of G that satisfies $\mu(v) \in L(v)$ for every $v \in V(G)$. In the q-List Coloring problem we are given a graph G together with a list $L(v) \subseteq [q]$ for every vertex v. The task is to determine whether there exists a proper list coloring of G.

A *feedback vertex set* of a graph G is a set $S \subseteq V(G)$ such that $G \setminus S$ is a forest; we denote by $\mathrm{fvs}(G)$ the size of the smallest such set. It is well-known that $\mathrm{tw}(G) \leq \mathrm{fvs}(G) + 1$. Unlike the other sections, where we give lower bounds for algorithms parameterized by $\mathrm{pw}(G)$, the following theorem gives also a lower bound for algorithms parameterized by $\mathrm{fvs}(G)$. Such a lower bound follows very naturally from the construction we are doing here, but not from the constructions in the other sections. It would be interesting to explore whether it is possible to prove tight bounds parameterized by $\mathrm{fvs}(G)$ for the problems considered in the other sections.

Theorem 2.16. *Let q be a fixed positive integer. If q-Coloring can be solved in $(q-\epsilon)^{\mathrm{fvs}(G)} \cdot n^{O(1)}$ or $(q-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ time for some $\epsilon > 0$, then* SAT *can be solved in $(2-\delta)^n \cdot n^{O(1)}$ time for some $\delta > 0$.*

Construction. We will show the result for List Coloring first, and then give a simple reduction that demonstrates that q-Coloring can be solved in $(q-\epsilon)^{\mathrm{fvs}(G)} \cdot n^{O(1)}$ time if and only if q-List Coloring can.

Depending on ϵ and q we choose a parameter p. Now, given an instance ϕ to SAT we will construct a graph G with a list $L(v)$ for every v, such that G has a proper list-coloring if and only if ϕ is satisfiable. Throughout the construction we will call color 1-*red*, color 2-*white* and color 3-*black*.

We start by grouping the variables of ϕ into t groups $F_1, \ldots, F_t$ of size at most $\lfloor \log q^p \rfloor$. Thus $t = \lceil \frac{n}{\lfloor \log q^p \rfloor} \rceil$. We will call an assignment of truth values to the variables in a group F_i a *group assignment*. We will say that a group assignment satisfies a clause C_j of ϕ if C_j contains at least one literal which is set to true by the group assignment. Notice that C_j can be satisfied by a group assignment of a group F_i, even though C_j also contains variables that are not in F_i.

For each group F_i, we introduce a set V_i of p vertices $v_i^1, \ldots, v_i^p$. The vertices in V_i get full lists, that is, they can be colored by any color in $[q]$. The coloring of the vertices in V_i will encode the group assignment of F_i. There are $q^p \geq 2^{|F_i|}$ possible colorings of V_i. Thus, to each possible group assignment of F_i we attach a unique coloring of V_i. Notice that some colorings of V_i may not correspond to any group assignments of F_i.

For each clause C_j of ϕ, we introduce a gadget $\widehat{C}_j$. The main part of $\widehat{C}_j$ is a long path $\widehat{P}_j$ that has one vertex for each group assignment that satisfies $\widehat{C}_j$. Notice that there are at most tq^p possible group assignments, and that q and p are constants independent of the input ϕ. The list of every

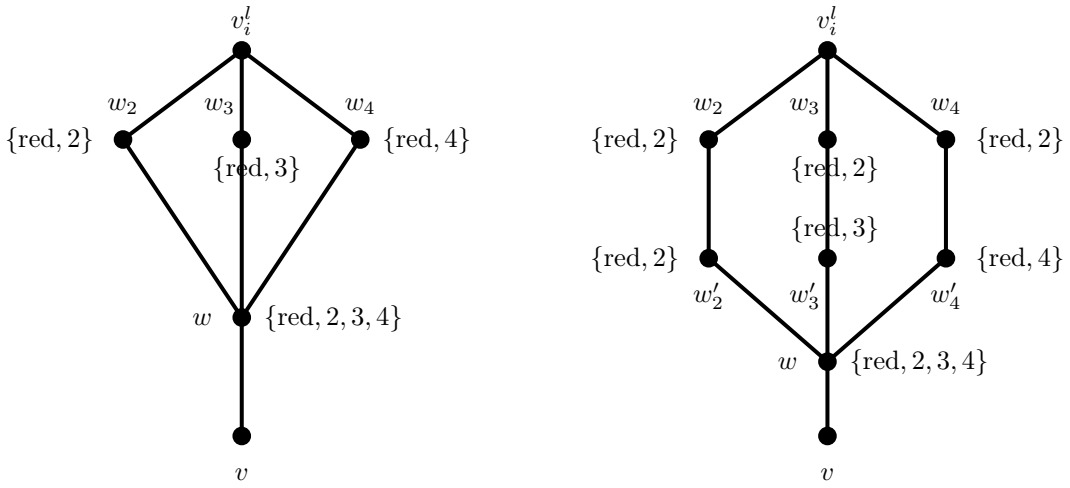

Figure 2.5: Reduction to q-Coloring: the way the connector connects a vertex v_i^ℓ with v for a particular "bad color" $x \in [q] \setminus \{\mu_i(v_i^\ell)\}$. The left side shows the case $x = \text{red} = 1$, the right side $x = 2$ ($q = 4$).

vertex on $\widehat{P}_j$ is $\{\text{red}, \text{white}, \text{black}\}$. We attach two vertices p_j^{start} and p_j^{end} to the start and end of $\widehat{P}_j$ respectively, and the two vertices are not counted as vertices of the path $\widehat{P}_j$ itself. The list of p_j^{start} is $\{\text{white}\}$. If $|V(\widehat{P}_j)|$ is even, then the list of p_j^{end} is $\{\text{white}\}$, whereas if $|V(\widehat{P}_j)|$ is odd then the list of p_j^{end} is $\{\text{black}\}$. The intention is that to properly color $\widehat{P}_j$, one needs to use the color red at least once, and that once is sufficient. The position of the red-colored vertex on the path $\widehat{P}_j$ encodes how the clause C_j is satisfied.

For every vertex v on $\widehat{P}_j$, we proceed as follows. The vertex v corresponds to some group assignment to F_i that satisfies the clause C_j. This assignment in turn corresponds to a coloring of the vertices of V_i. Let this coloring be μ_i. We build a *connector* whose role is to enforce that v can be red only if coloring μ_i appears on V_i. To build the connector, for each vertex $v_i^\ell \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^\ell)\}$ we do the following to enforce that if v is red, then v_i^ℓ *cannot* have color x (see Figure 2.5).

- If x is red, then we introduce one vertex w_y for every color y except for red. We make w_y adjacent to v_i^ℓ and the list of w_y is $\{\text{red}, y\}$. Then we introduce a vertex w that is adjacent to v and to all vertices w_y. The list of w is all of $[q]$.
- If x is not red, we introduce two vertices w_y and w_y' for each color y except for red. We make w_y adjacent to v_i^ℓ and w_y' adjacent to w_y. The list of w_y is $\{x, \text{red}\}$ while the list of w_y' is $\{y, \text{red}\}$. Finally, we introduce a vertex w adjacent to v and to w_y' for all y. The list of w is all of $[q]$.

Notice that in the above construction we have reused the names w, w_y and w_y' for many different vertices: in each connector, there is a separate vertex w for each vertex $v_i^\ell \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^\ell)\}$. Building a connector for each vertex v on $\widehat{P}_j$ concludes the construction of the clause gadget $\widehat{C}_j$, and creating one such gadget for each clause concludes the construction of G (see Figure 2.6). The following lemma, summarizes the most important properties of the connector:

Lemma 2.17. *Consider the connector corresponding to a vertex v on $\widehat{P}_j$ and a coloring μ_i of V_i.*

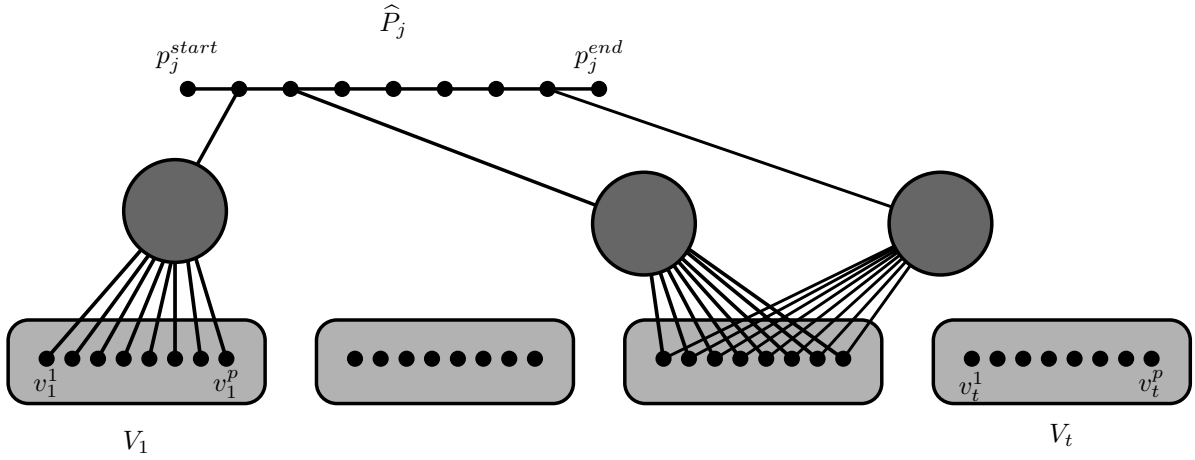

Figure 2.6: Reduction to q-Coloring. The t groups of vertices $V_1, \ldots, V_t$ represent the t groups of variables $F_1, \ldots, F_t$ (each of size $\lceil \log q^p \rceil$). Each vertex of the clause path $\widehat{P}_j$ is connected to one group V_i via a connector (multiple vertices on the path can be connected to the same group).

1. *Any coloring on V_i and any color $c \in \{\text{white}, \text{black}\}$ on v can be extended to the rest of the connector.*
2. *Coloring μ_i on V_i and any color $c \in \{\text{red}, \text{white}, \text{black}\}$ on v can be extended to the rest of the connector.*
3. *In any coloring of the connector, if v is red, then μ_i appears on V_i.*

Proof. 1. For each vertex $v_i^\ell \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^\ell)\}$ we do the following.

- If x is red, then in the construction of $\widehat{C}_j$ we introduced a vertex w_y with list $\{y, \text{red}\}$ for every color $y \neq$ red adjacent to v_i^ℓ, and a vertex w with list $[q]$ adjacent to w_y for every $y \neq$ red. If v_i^ℓ is colored red, then we color each vertex w_y with y and w with red. Notice that w is adjacent to v, but v is colored either white or black, so it is safe to color w red. If, on the other hand, v_i^ℓ is not colored red, we can color w_y red for every y. Then all the neighbours of w have been colored with red, except for v which has been colored white or black. Thus it is safe to color w with the color out of black and white which was not used to color v.

- If x is not red, then in the construction of $\widehat{C}_j$ we introduced two vertices w_y and w'_y for each color y except for red, and also introduced a vertex w. The vertices w_y are adjacent to v_i^ℓ and for every $y \neq$ red, the vertex w'_y is adjacent to w_y. Finally, w is adjacent to all the vertices w'_y and to v. For every y the list of w_y is $\{x, \text{red}\}$ while the list of w'_y is $\{y, \text{red}\}$. The list of w is $[q]$. If v_i^ℓ is colored with x, then we let w_y take color red and w'_y take color y for every $y \neq$ red. We color w with red. In the case that v_i^ℓ is colored with a color different from x, we let w_y be colored with x and w'_y be colored red for every $y \neq$ red. Finally, all the neighours of w except for v have been colored red, while v is colored with either black or white. According to the color of v, we can either color w black or white.

2. We can assume that v is red, otherwise we are done by the previous statement. For each vertex $v_i^\ell \in V_i$ and color $x \in [q] \setminus \{\mu_i(v_i^\ell)\}$, we do the following.

- If x is red, then in the construction of $\widehat{C}_j$ we introduced a vertex w_y with list $\{y, \text{red}\}$ for every color $y \neq$ red adjacent to v_i^ℓ, and a vertex w with list $[q]$ adjacent to w_y for every $y \neq$ red.

Since $v_{i'}^{\ell}$ is not colored red by μ_i, we can color w_y red for every y. Then all the neighbours of w including v have been colored with red and it is safe to color w with white.

- If x is not red, then in the construction of $\widehat{C}_j$ we introduced two vertices w_y and w'_y for each color y except for red, and also introduced a vertex w. The vertices w_y are adjacent to v_i^{ℓ} and for every $y \neq$ red the vertex w'_y is adjacent to w_y. Finally, w is adjacent to all the vertices w'_y and to v. For every y, the list of w_y is $\{x, \text{red}\}$ while the list of w'_y is $\{y, \text{red}\}$. The list of w is $[q]$. Since μ_i colors v_i^{ℓ} with a color different from x, we let w_y be colored with x and w'_y be colored red for every $y \neq$ red. Finally, all the neighours of w including v have been colored red so it is safe to color w white.

3. Suppose for contradiction that v is red, but some vertex $v_i^{\ell} \in V_i$ has been colored with a color $x \neq \mu_i(v_i^{\ell})$. There are two cases. If x is red, then in the construction we introduced vertices w_y adjacent to v_i^{ℓ} for every color $y \neq$ red. Also we introduced a vertex w adjacent to v and to w_y for each $y \neq$ red. The list of w_y is $\{\text{red}, y\}$ and hence w_y must have been colored y for every $y \neq$ red. But then w is adjacent to v which is colored red, and to w_y which is colored y for every $y \neq$ red. Thus vertex w has all colors in its neighborhood, a contradiction. In the case when x is not red, then in the construction we introduced two vertices w_y and w'_y for each $y \neq$ red. Each w_y was adjacent to v_i^{ℓ} and had $\{x, \text{red}\}$ as its list. Since v_i^{ℓ} is colored x, all the w_y vertices must be colored red. For every $y \neq$ red, we have that w'_y is adjacent to w_y and has $\{\text{red}, y\}$ as its list. Hence for every $y \neq$ red, the vertex w'_y is colored with y. But, in the construction we also introduced a vertex w adjacent to v and to w'_y for each $y \neq$ red. Thus again, vertex w has all colors in its neighbourhood, a contradiction. □

Lemma 2.18. *If ϕ is satisfiable, then G has a proper list-coloring.*

Proof. Starting from a satisfying assignment of ϕ, we construct a coloring γ of G. The assignment to ϕ corresponds to a group assignment to each group F_i. Each group assignment corresponds to a coloring of V_i. For every i, we let γ color the vertices of V_i using the coloring corresponding to the group assignment of F_i.

Now we show how to complete this coloring to a proper coloring of G. Since the gadgets $\widehat{C}_j$ are pairwise disjoint, and there are no edges going between them, it is sufficient to show that we can complete the coloring for every gadget $\widehat{C}_j$. Consider the clause C_j. The clause contains a literal that is set to true, and this literal belongs to a variable in some group F_i. The group assignment of F_i satisfies the clause C_j. Thus, there is a vertex v on $\widehat{P}_j$ that corresponds to this assignment. We set $\gamma(v)$ as red (that is, γ colors v red), p_j^{start} is colored white and p_j^{end} is colored with its only admissible color, namely black if $|V(\widehat{P}_j)|$ is even and white if $|V(\widehat{P}_j)|$ is odd. The remaining vertices of $\widehat{P}_j$ are colored alternatingly white or black. By Lemma 2.17(2), the coloring can be extended to every vertex of the connector between V_i and v: the coloring appearing on V_i is the coloring μ_i corresponding to the group assignment F_i. For every other vertex u on $\widehat{P}_j$, the color of u is black or white, thus Lemma 2.17(1) ensures that the coloring can be extended to any connector on u.

As this procedure can be repeated to color the gadget $\widehat{C}_j$ for every clause C_j, we can complete γ to a proper list-coloring of G. □

Lemma 2.19. *If G has a proper list-coloring γ, then ϕ is satisfiable.*

Proof. Given γ, we construct an assignment to the variables of ϕ as follows. For every group F_i of variables, if γ colors V_i with a coloring that corresponds to a group assignment of F_i, then we set this assignment for the variables in F_i. Otherwise, we set all the variables in F_i to false. We need to argue that this assignment satisfies all the clauses of ϕ.

Consider a clause C_j and the corresponding gadget $\widehat{C}_j$. By a simple parity argument, $\widehat{P}_j$ cannot be colored using only the colors black and white. Thus, some vertex v on $\widehat{P}_j$ is colored red. The vertex v corresponds to a group assignment of some group F_i that satisfies $\widehat{C}_j$. As v is red, Lemma 2.17(3) implies that V_i is colored with the coloring μ_i that corresponds to this assignment. The construction then implies that our chosen assignment satisfies C_j. As this is true for every clause, this concludes the proof. □

Observation 2.20. *The vertices $\bigcup_{i\leq t} V_i$ form a feedback vertex set of G. Furthermore,* $\mathrm{pw}(G) \leq pt + 4$

Proof. Observe that after removing $\bigcup_{i\leq t} V_i$, all that is left are the gadgets $\widehat{C}_j$, which do not have any edges between each other. Each such gadget is a tree and hence $\bigcup_{i\leq t} V_i$ form a feedback vertex set of G. If we place a searcher on each vertex of $\bigcup_{i\leq t} V_i$ it is easy to see that each gadget $\widehat{C}_j$ can be searched with 4 searchers. The pathwidth bound on G follows using Proposition 2.2. □

Lemma 2.21. *If q-*List Coloring *can be solved in $(q-\epsilon)^{\mathrm{fvs}(G)} \cdot n^{O(1)}$ or $(q-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ time for some $\epsilon > 0$, then* SAT *can be solved in $(2-\delta)^n \cdot n^{O(1)}$ time for some $\delta > 0$.*

Proof. Let $(q-\epsilon)^{\mathrm{fvs}(G)} \cdot n^{O(1)} = O^*(q^{\lambda \mathrm{fvs}(G)})$ time, where $\lambda = \log_q(q-\epsilon) < 1$. We choose a sufficiently large p such that $\delta' = \lambda\frac{p}{p-1} < 1$. Given an instance ϕ of SAT, we construct a graph G using the construction above, and run the assumed q-List Coloring. Correctness follows from Lemmata 2.18 and 2.19. By Observation 2.20, the graph G has a feedback vertex set of size $p\lceil\frac{n}{\lfloor p \log q\rfloor}\rceil$. The choice of p implies that

$$\lambda p\lceil\frac{n}{\lfloor p\log q\rfloor}\rceil \leq \lambda p\frac{n}{(p-1)\log q} + p \leq \delta'\frac{n}{\log q} + p \leq \delta'' n,$$

for some $\delta'' < 1$. Hence SAT can be solved in time $2^{\delta'' n} \cdot n^{O(1)} = (2-\delta)^n \cdot n^{O(1)}$, for some $\delta > 0$. By Observation 2.20, we also know that $\mathrm{pw}(G) \leq pt + 4$. Thus, the feedback vertex set size and the pathwidth of the constructed graph just differs by 4. This implies that q-List Coloring cannot be solved in $(q-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ time. □

Finally, observe that we can reduce q-List-Coloring to q-Coloring by adding a clique $Q = \{q_1, \ldots, q_c\}$ on q vertices to G and making q_i adjacent to v when $i \notin L(v)$. Any coloring of Q must use q different colors, and without loss of generality q_i is colored with color i. Then one can complete the coloring if and only if one can properly color G using a color from $L(v)$ for each v. We can add the clique Q to the feedback vertex set—this increases the size of the minimum feedback vertex set by q. Since q is a constant independent of the input, this yields Theorem 2.16.

2.6 Odd Cycle Transversal

An equivalent formulation of the Max Cut problem is to ask for a bipartite subgraph with the *maximum* number of edges, which is the same as asking for a set of edges of *minimum* size whose deletion makes the graph bipartite. We can also consider the vertex-deletion version of this problem. An *odd cycle transversal* of a graph G is a subset $S \subseteq V(G)$ such that $G \setminus S$ is bipartite. In the Odd Cycle Transversal problem, we are given a graph G together with an integer k and asked whether G has an odd cycle transversal of size k.

Theorem 2.22. *If* Odd Cycle Transversal *can be solved in $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ time for $\epsilon > 0$, then* SAT *can be solved in $(2-\delta)^n \cdot n^{O(1)}$ time for some $\delta > 0$.*

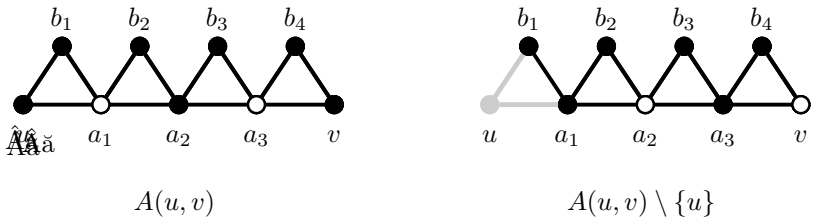

Figure 2.7: Reduction to Odd Cycle Transversal. The arrow $A(u,v)$ from u to v with the passive odd cycle transversal shown in white (left) and the active odd cycle transversal of $A(u,v)\setminus\{u\}$ (right).

Construction. Given $\epsilon > 0$ and an instance ϕ of SAT, we construct a graph G as follows. We choose an integer p based just on ϵ. Exactly how p is chosen will be discussed at the end of this section. We start by grouping the variables of ϕ into t groups $F_1, \ldots, F_t$ of size at most $h = \lfloor \log 3^p \rfloor$. Thus $t = \lceil \frac{n}{\lfloor \log 3^p \rfloor} \rceil$. We will call an assignment of truth values to the variables in a group F_i a *group assignment*. We will say that a group assignment satisfies a clause C_j of ϕ if C_j contains at least one literal that is set to true by the group assignment. Notice that C_j can be satisfied by a group assignment of a group F_i even though C_j also contains variables that are not in F_i.

Now we describe an auxiliary gadget which will be very useful in our construction (see Figure 2.7). For two vertices u and v by *adding an arrow* from u to v we will mean adding a path $ua_1a_2a_3v$ on four edges starting in u and ending in v. Furthermore, we add four vertices b_1, b_2, b_3 and b_4 and edges ub_1, b_1a_1, a_1b_2, b_2a_2, a_2b_3, b_3a_3, a_3b_4, b_4v, and b_4v. Denote the resulting graph $A(u,v)$. None of the vertices in $A(u,v)$ except for u and v will receive any further neighbours throughout the construction of G. The graph $A(u,v)$ has the following properties, which are useful for our construction.

- The unique smallest odd cycle transversal of $A(u,v)$ is $\{a_1, a_3\}$. We call this the *passive* odd cycle transversal of the arrow.
- In $A(u,v) \setminus \{a_1, a_3\}$, u and v are in different connected components.
- The set $\{a_2, v\}$ is a smallest odd cycle transversal of $A(u,v) \setminus \{u\}$. We call this the *active* odd cycle transversal of the arrow.

The intuition behind an arrow from u to v is that if u is put into the odd cycle transversal, then v can be put into the odd cycle transversal "for free." When the active odd cycle transversal of the arrow is picked, we say the arrow is active, otherwise we say the arrow is passive.

To construct G, we make $t \cdot p$ paths, $\{P_{i,j}\}$ for $1 \le i \le t$, $1 \le j \le p$ (see Figure 2.8). Each path has $3m(tp+1)$ vertices, and the vertices of $P_{i,j}$ are denoted by $p^\ell_{i,j}$ for $1 \le \ell \le 3m(tp+1)$. For a fixed i, the paths $\{P_{i,j} : 1 \le j \le p\}$ correspond to the set F_i of variables. For every $1 \le i \le t$, $1 \le j \le p$ and $1 \le \ell < 3m(tp+1)$ we add three vertices $a^\ell_{i,j}$, $b^\ell_{i,j}$ and $q^\ell_{i,j}$ adjacent to each other. We also add the edges $a^\ell_{i,j}p^\ell_{i,j}$ and $b^\ell_{i,j}p^{\ell+1}_{i,j}$. One can think of the vertices of the paths $\{P_{i,j}\}$ layed out as rows in a matrix, where for every fixed $1 \le \ell \le 3m(tp+1)$ there is a column $\{p^\ell_{i,j} : 1 \le i \le t, 1 \le j \le p\}$. We group the colums three by three. In particular, For every $i \le t$ and $0 \le \ell < m(tp+1)$ we define the sets $P^\ell_i = \{p^{3\ell+1}_{i,j}, p^{3\ell+2}_{i,j}, p^{3\ell+3}_{i,j} : 1 \le j \le p\}$, $A^\ell_i = \{a^{3\ell+1}_{i,j}, a^{3\ell+2}_{i,j}, a^{3\ell+3}_{i,j} : 1 \le j \le p\}$, $B^\ell_i = \{b^{3\ell+1}_{i,j}, b^{3\ell+2}_{i,j}, b^{3\ell+3}_{i,j} : 1 \le j \le p\}$ and $Q^\ell_i = \{q^{3\ell+1}_{i,j}, q^{3\ell+2}_{i,j}, q^{3\ell+3}_{i,j} : 1 \le j \le p\}$.

For every $i \le t$ and $0 \le \ell < m(tp+1)$ we make two new sets L^ℓ_i and R^ℓ_i of new vertices. Both L^ℓ_i and R^ℓ_i are independent sets of size $5p$, and we add all the edges possible between L^ℓ_i and R^ℓ_i. From L^ℓ_i we pick a special vertex λ^ℓ_i and from R^ℓ_i we pick ρ^ℓ_i. We make all the vertices in A^ℓ_i adjacent to

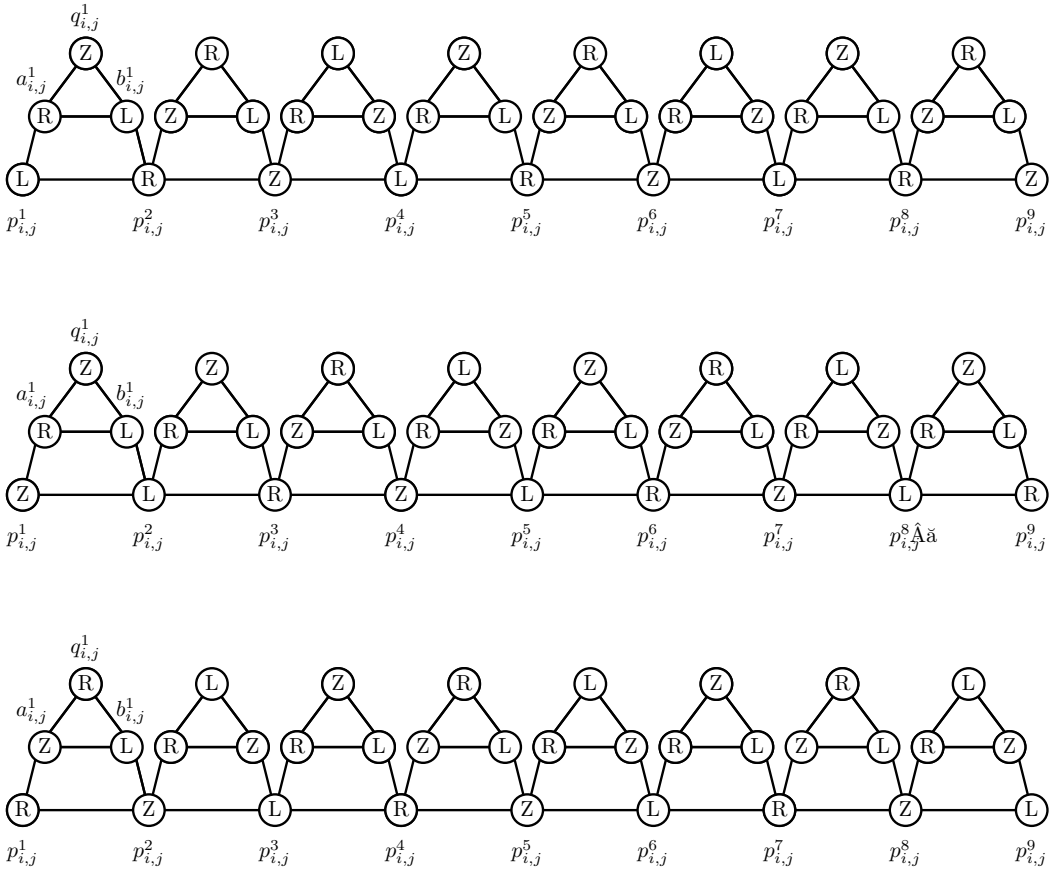

Figure 2.8: Reduction to ODD CYCLE TRANSVERSAL. The path $P_{i,j}$ with three different ways of removing a set Z and partitioning the remaining bipartite graph into classes L and R.

all vertices of L^ℓ_i, and we make all vertices in B^ℓ_i adjacent to all vertices of R^ℓ_i. We make λ^ℓ_i adjacent to $\rho^{\ell+1}_i$, except for $\ell = m(tp+1)-1$.

We will say that a subset S of P^ℓ_i which picks exactly one vertex from $P_{i,j}$ for every $1 \leq j \leq p$ is *good*. The idea is that there are $3^p \geq 2^h$ good subsets of P^ℓ_i, so we can make group assignments of F_i correspond to good subsets of P^ℓ_i. For every good subset S of P^ℓ_i we add a cycle $X^\ell_{i,S}$. The cycle $X^\ell_{i,S}$ has length $2p+1$. We select a vertex on $X^\ell_{i,S}$ and call it $x^\ell_{i,S}$. For every vertex $u \in P^\ell_i \setminus S$ we add an arrow from u to a vertex of $X^\ell_{i,S}$. We add arrows in such a way that every vertex of $X^\ell_{i,S} \setminus \{x^\ell_{i,S}\}$ is the endpoint of exactly one arrow.

For every $i \leq t$ and $0 \leq \ell < m(tp+1)$, we make a cycle Y^ℓ_i of length 3^p.Notice that the length of the cycle is odd. Every vertex of Y^ℓ_i corresponds to a good subset S of P^ℓ_i. For each good subset S of P^ℓ_i we add an arrow from $x^\ell_{i,S}$ of the cycle $X^\ell_{i,S}$ to the vertex in Y^ℓ_i that corresponds to S.

We say that a good subset of P^ℓ_i is *equal* with a good subset S' of $P^{\ell'}_i$ if for every $1 \leq j \leq t$, the distance along $P_{i,j}$ between the vertex of S on $P_{i,j}$ and the vertex of S' on $P_{i,j}$ is divisible by 3. Informally, S and S' are equal if they look identical when we superimpose P^ℓ_i onto $P^{\ell'}_i$. To every group assignment of variables F_i, we designate a good subset of P^ℓ_i for every ℓ. We designate good subsets in such a way that good subsets corresponding to the same group assignment are equal.

Finally, for every clause C_j, $1 \leq j \leq m$, we will introduce $tp+1$ cycles. That is, for every $0 \leq r \leq tp$, we inroduce a cycle $\widehat{C}^r_j$. The cycle contains one vertex for every $i \leq t$ and group

assignment to F_i, and potentially one dummy vertex to make it have odd length. Going around the cycle counterclockwise we first encounter all the vertices corresponding to group assignments of F_1, then all the vertices corresponding to group assignments of F_2, and so on. For $i \leq t$ and every good subset S of P_i^{rm+j} that corresponds to a group assignment of F_i that satisfies C_j we add an arrow from $x_{i,S}^{rm+j}$ to the vertex on $\widehat{C}_j^r$ that corresponds to the same group assignment of F_i as S does. This concludes the construction of G.

The intention behind the construction is that if ϕ is satisfiable, then a minimum odd cycle transversal of G can pick:

- One vertex from each triangle $\{a_{i,j}^\ell, b_{i,j}^\ell, q_{i,j}^\ell\}$ for each $1 \leq i \leq t$, $1 \leq j \leq p$, $1 \leq \ell < 3m(tp+1)$. There are $tp(3m(tp+1)-1)$ such triangles in total.
- One vertex from $\{p_{i,j}^{3\ell+1}, p_{i,j}^{3\ell+2}, p_{i,j}^{3\ell+3}\}$ for each $1 \leq i \leq t$, $1 \leq j \leq p$, $0 \leq \ell < m(tp+1)$. There are $tpm(tp+1)$ such triples.
- Two vertices from every arrow added, *without* counting the starting point of the arrow. For each $i \leq t$ and $0 \leq \ell < m(tp+1)$, there are $2p3^p$ arrows ending in some cycle $X_{i,S}^\ell$. Hence there are $2p3^p tm(tp+1)$ such arrows. For every $i \leq t$ and $0 \leq \ell < m(tp+1)$ there are 3^p arrows ending in the cycle Y_i^ℓ. Hence there are $3^p tm(tp+1)$ such arrows. For every clause C_j, there are $tp+1$ arrows added for every group assignment that satisfies that clause. Let μ be the sum over all clauses of the number of group assignments that satisfy that clause. The total number of arrows added is then $\mu(tp+1) + (2p+1)3^p tm(tp+1)$. Thus the odd cycle transversal can pick $2\mu(tp+1) + 2(2p+1)3^p tm(tp+1)$ vertices from arrows.
- One vertex $x_{i,S}^\ell$ for every $i \leq t$ and $0 \leq \ell < m(tp+1)$. There are $tm(tp+1)$ choices for i and ℓ.

We let the α be the value of the total budget, that is the sum of the items above.

Lemma 2.23. *If ϕ is satisfiable, then G has an odd cycle transversal of size α.*

Proof. Given a satisfying assignment γ to ϕ, we construct an odd cycle transversal Z of G of size α together with a partition of $V(G) \setminus Z$ into L and R such that every edge of $G \setminus Z$ goes between a vertex in L and a vertex in R. The assignment to ϕ corresponds to a group assignment of each F_i for $1 \leq i \leq t$. For every $1 \leq i \leq t$ and $0 \leq \ell < m(tp+1)$, we add to Z the good subset S of P_i^ℓ that corresponds to the group assignment of F_i. Notice that for each fixed i, the sets picked from P_i^ℓ and $P_i^{\ell'}$ are equal for any ℓ, ℓ'. At this point we have picked one vertex from $\{p_{i,j}^{3\ell+1}, p_{i,j}^{3\ell+2}, p_{i,j}^{3\ell+3}\}$ for each $1 \leq i \leq t$, $1 \leq j \leq p$, $0 \leq \ell < m(tp+1)$.

For every fixed $1 \leq i \leq t$, $1 \leq j \leq p$, there are three cases. If $p_{i,j}^1 \in Z$, we put $p_{i,j}^2$ into L and $p_{i,j}^3$ into R. If $p_{i,j}^2 \in Z$ we put $p_{i,j}^1$ into R and $p_{i,j}^3$ into L. If $p_{i,j}^3 \in Z$ we put $p_{i,j}^1$ into L and $p_{i,j}^2$ into R. Now, for every $4 \leq \ell \leq 3m(tp+1)$ such that $p_{i,j}^\ell \notin Z$ we put $p_{i,j}^\ell$ into the same set out of $\{L, R\}$ as $p_{i,j}^{\ell'}$ where $1 \leq \ell' \leq 3$ and $\ell \equiv \ell' \mod 3$.

For every $1 \leq i \leq t$, $0 \leq \ell \leq m(tp+1)$, we put L_i^ℓ into L and R_i^ℓ into R. For every triple of a, b, q of pairwise adjacent vertices such that $a \in A_i^\ell$, $b \in B_i^\ell$, and $q \in Q_i^\ell$, we proceed as follows. The vertex a has a neighbour a' in P_i^ℓ and b has a neighbour b' in P_i^ℓ. There is a j such that b' is the successor of a' on $P_{i,j}$. Thus, there are three cases;

- $a' \in Z$ and $b' \in L$, we put a in R, q in L and b in Z.
- $a' \in R$ and $b' \in Z$, we put a in Z, q in R and b in L.
- $a' \in L$ and $b' \in R$, we put a in R, q in Z and b in L.

For every $1 \leq i \leq t$, $0 \leq \ell \leq m(tp+1)$, there are many arrows from vertices in P_i^ℓ to vertices on cycles $X_{i,S}^\ell$ for good subsets S of P_i^ℓ. For each arrow, if its endpoint in P_i^ℓ is in Z we add the active

odd cycle transversal of the arrow to Z, otherwise we add the passive odd cycle transversal of the arrow to Z. In either case, the remaining vertices on the arrow form a forest, and therefore we can insert the remaining vertices of the arrow into L and R according to which sets out of $\{L, R, Z\}$ u and v are in.

For every $1 \leq i \leq t$, $0 \leq \ell \leq m(tp+1)$, there is exactly one set S such that the cycle $X^\ell_{i,S}$ only has passive arrows pointing into it. This is exactly the set S which corresponds to the restriction of γ to F_i. Each cycle $X^\ell_{i,S'}$ that has at least one arrow pointing into them already contains at least one vertex in Z—the endpoint of the active arrow pointing into the cycle. Thus we can partition the remaining vertices of $X^\ell_{i,S'}$ into L and R such that no edge has both endpoints in L or both endpoints in R. For the cycle $X^\ell_{i,S}$, we put $x^\ell_{i,S}$ into Z and partition the remaining vertices of $X^\ell_{i,S}$ into L and R such that no edge has both endpoints in L or both endpoints in R. We add the active odd cycle transversal in the arrow from $x^\ell_{i,S}$ to the cycle Y^ℓ_i into Z. For all other good subsets S', we add the passive odd cycle transversal in the arrow from $x^\ell_{i,S}$ to the cycle Y^ℓ_i into Z. Thus each cycle Y^ℓ_i contains one vertex in Z and the remaining vertices of Y^ℓ_i can be distributed into L and R.

For every arrow that goes from a vertex $x^\ell_{i,S}$ into a cycle $\widehat{C}^r_h$, we add the active odd cycle transversal of the arrow to Z if $x^\ell_{i,S} \in Z$ and add the passive odd cycle transversal to Z otherwise. Again the remaining vertices on each arrow can easily be partitioned into L and R such that no edge has both endpoints in L or both endpoints in R. This concludes the construction of Z. Since we have put the vertices into Z in accordance to the budget described in the construction it follows that $|Z| \leq \alpha$. All that remains to show, is that for each $1 \leq h \leq m$ and $0 \leq r \leq tp$, the cycle $\widehat{C}^r_h$ has at least one active arrow pointing into it.

The cycle $\widehat{C}^r_h$ corresponds to the clause C_h. The clause C_h is satisfied by γ and hence it is satisfied by the restriction of γ to some group F_i. This restriction is a group assignment of F_i and hence it corresponds to a good subset S of P^{rm+h}_i, which happens to be exactly $Z \cap P^{rm+h}_i$. Thus $x^{rm+h}_{i,S} \in Z$ and since the restriction of γ to F_i satisfies C_h, there is an arrow pointing from $x^{rm+h}_{i,S}$ and into $\widehat{C}^r_h$. Since this arrow is active, this concludes the proof. □

Lemma 2.24. *If G has an odd cycle transversal of size α, then ϕ is satisfiable.*

Proof. Let Z be an odd cycle transversal of G of size α. Since $G \setminus Z$ is bipartite, the vertices of $G \setminus Z$ can be partitioned into L and R such that every edge of $G \setminus Z$ has one endpoint in L and the other in R. Given Z, L and R, we construct a satisfying assignment to ϕ. Every arrow in G must contain at least two vertices in Z, not counting the startpoint of the arrow. Let $\vec{Z}$ be a subset of Z containing two vertices from each arrow, but no arrow start point. Observe that no two arrows have the same endpoint, and therefore $|\vec{Z}|$ is exactly two times the number of arrows in G. Let $Z' = Z \setminus \vec{Z}$.

We argue that for any $1 \leq i \leq t$ and $0 \leq \ell < m(tp+1)$ we have $|Z' \cap (L^\ell_i \cup R^\ell_i \cup A^\ell_i \cup B^\ell_i \cup Q^\ell_i \cup P^\ell_i)| \geq 4p$. Observe that no vertices in L^ℓ_i, R^ℓ_i, A^ℓ_i, B^ℓ_i, Q^ℓ_i or P^ℓ_i are endpoints of arrows, and hence they do not contain any vertices of $\vec{Z}$. Suppose for contradiction that $|Z' \cap (L^\ell_i \cup R^\ell_i \cup A^\ell_i \cup B^\ell_i \cup Q^\ell_i \cup P^\ell_i)| < 4p$. Then there is a vertex in $\lambda \in L^\ell_i \setminus Z'$, and a vertex $\rho \in R^\ell_i \setminus Z'$. Without loss of generality, $\lambda \in L$ and $\rho \in R$. Furthermore, there is a $1 \leq j \leq p$ such that

$$|Z' \cap \{p^{3\ell+1}_{i,j}, p^{3\ell+2}_{i,j}, p^{3\ell+3}_{i,j}, a^{3\ell+1}_{i,j}, a^{3\ell+2}_{i,j}, a^{3\ell+3}_{i,j}, b^{3\ell+1}_{i,j}, b^{3\ell+2}_{i,j}, b^{3\ell+3}_{i,j}, q^{3\ell+1}_{i,j}, q^{3\ell+2}_{i,j}, q^{3\ell+3}_{i,j}\}| < 4.$$

Since $\{a^{3\ell+1}_{i,j}, b^{3\ell+1}_{i,j}, q^{3\ell+1}_{i,j}\}$, $\{a^{3\ell+2}_{i,j}, b^{3\ell+2}_{i,j}, q^{3\ell+2}_{i,j}\}$ and $\{a^{3\ell+3}_{i,j}, b^{3\ell+3}_{i,j}, q^{3\ell+3}_{i,j}\}$ form triangles and must contain a vertex from Z' each, it follows that each of these triangles contain exactly one vertex from

Z', and that $Z' \cap \{p_{i,j}^{3\ell+1}, p_{i,j}^{3\ell+2}, p_{i,j}^{3\ell+3}\} = \emptyset$. Since $\lambda \in L$ and $\rho \in R$, λ is adjacent to all vertices of $A_{i,j}^{\ell}$ and ρ is adjacent to all vertices of $B_{i,j}^{\ell}$, it follows that $A_{i,j}^{\ell} \setminus Z' \subseteq R$ and $B_{i,j}^{\ell} \setminus Z' \subseteq L$.

Hence, there are two cases to consider either (1) $\{p_{i,j}^{3\ell+1}, p_{i,j}^{3\ell+3}\} \subseteq L$ and $p_{i,j}^{3\ell+2} \in R$ or (2) $\{p_{i,j}^{3\ell+1}, p_{i,j}^{3\ell+3}\} \subseteq R$ and $p_{i,j}^{3\ell+2} \in L$. In the first case, observe that either $a_{i,j}^{3\ell+2} \in R$ or $b_{i,j}^{3\ell+2} \in L$ and hence either $a_{i,j}^{3\ell+2} p_{i,j}^{3\ell+2}$ or $b_{i,j}^{3\ell+2} p_{i,j}^{3\ell+3}$ have both endpoints in the same set out of $\{L, R\}$, a contradiction. The second case is similar, either $a_{i,j}^{3\ell+1} \in R$ or $b_{i,j}^{3\ell+1} \in L$ and hence either $a_{i,j}^{3\ell+1} p_{i,j}^{3\ell+1}$ or $b_{i,j}^{3\ell+1} p_{i,j}^{3\ell+2}$ have both endpoints in the same set out of $\{L, R\}$, a contradiction. We conclude that $|Z' \cap (L_i^{\ell} \cup R_i^{\ell} \cup A_i^{\ell} \cup B_i^{\ell} \cup Q_i^{\ell} \cup P_i^{\ell})| \geq 4p$.

For any $1 \leq i \leq t$ and $0 \leq \ell < m(tp+1)$, Y_i^{ℓ} is an odd cycle so Y_i^{ℓ} contains a vertex in Z. If Y_i^{ℓ} contains no vertices of Z', then it contains a vertex from $\vec{Z}$ and there is an active arrow pointing into Y_i^{ℓ}. The starting point of this arrow is a vertex $x_{i,S}^{\ell}$ for some good subset S of P_i^{ℓ}. Since the arrow is active and $x_{i,S}^{\ell}$ is not the endpoint of any arrow, we know that $x_{i,S}^{\ell} \in Z'$. Hence for any $1 \leq i \leq t$ and $0 \leq \ell < m(tp+1)$, we have that either there is a good subset S of P_i^{ℓ} such that $x_{i,S}^{\ell} \in Z'$ or at least one vertex of Y_i^{ℓ} is in Z'.

The above arguments, together with the budget constraints, imply that for every $1 \leq i \leq t$ and $0 \leq \ell < m(tp+1)$, we have $|Z' \cap (L_i^{\ell} \cup R_i^{\ell} \cup A_i^{\ell} \cup B_i^{\ell} \cup Q_i^{\ell} \cup P_i^{\ell})| = 4p$ and that $|Z' \cap \bigcup \{x_{i,S}^{\ell}\} \cup V(Y_i^{\ell})| = 1$, where the union is taken over all good subsets S of P_i^{ℓ}. It follows $Z' \cap P_i^{\ell}$ is a good subset of P_i^{ℓ}. Let $S = Z' \cap P_i^{\ell}$. The cycle $X_{i,S}^{\ell}$ has odd length, and hence it must contain some vertex from Z. On the other hand, all the arrows pointing into $X_{i,S}^{\ell}$ are passive, so $X_{i,S}^{\ell}$ cannot contain any vertices from $\vec{Z}$. Thus $X_{i,S}^{\ell}$ contains a vertex from Z', and by the budget constraints this must be $x_{i,S}^{\ell}$.

Now, consider three consecutive vertices $p_{i,j}^{\ell}$, $p_{i,j}^{\ell+1}$, $p_{i,j}^{\ell+2}$ for some $1 \leq i \leq t$, $1 \leq j \leq p$, $1 \leq \ell \leq 3m(tp+1) - 2$. We prove that at least one of them has to be in Z. Suppose not. We know that neither $\lambda_i^{\lfloor \ell/3 \rfloor}$, $\rho_i^{\lfloor \ell/3 \rfloor}$, $\lambda_i^{\lfloor \ell/3 \rfloor + 1}$ nor $\rho_i^{\lfloor \ell/3 \rfloor + 1}$ are in Z. Thus, without loss of generality $\{\lambda_i^{\lfloor \ell/3 \rfloor}, \lambda_i^{\lfloor \ell/3 \rfloor + 1}\} \subseteq L$ and $\{\rho_i^{\lfloor \ell/3 \rfloor}, \rho_i^{\lfloor \ell/3 \rfloor + 1}\} \subseteq R$. There are two cases. Either $p_{i,j}^{\ell} \in R$ and $p_{i,j}^{\ell+1} \in L$ or $p_{i,j}^{\ell+1} \in L$ and $p_{i,j}^{\ell+3} \in R$. In the first case, we obtain a contradiction since either $a_{i,j}^{\ell} \in R$ or $b_{i,j}^{\ell} \in L$. In the second case, we get a contradiction since either $a_{i,j}^{\ell+1} \in R$ or $b_{i,j}^{\ell+1} \in L$. Hence for any three consecutive vertices on $P_{i,j}$, at least one of them is in Z. Since the budget constraints ensure that there are at most $|V(P_{i,j})|/3$ vertices in $P_{i,j} \cap Z$, it follows from the pigeon hole principle that there is an $0 \leq r \leq tp$ such that for any $1 \leq i \leq t$ and $1 \leq h \leq m$ and $1 \leq h' \leq m$, the set $P_i^{rm+h} \cap Z$ equals $P_i^{rm+h'} \cap Z$. Here equality is in the sense of equality of good subsets of P_i^{ℓ}.

For every $1 \leq i \leq t$, $P_i^{rm+1} \cap Z$ is a good subset of P_i^{rm+1}. If $P_i^{rm+1} \cap Z$ corresponds to a group assignment of F_i, then we set the variables in F_i to this assignment. Otherwise we set all the variables in F_i to false. We need to argue that every clause C_h is satisfied by this assignment. Consider the cycle $\widehat{C}_h^r$. Since it is an odd cycle, it must contain a vertex from Z, the budget constraints and the discussion above implies that this vertex is from $\vec{Z}$. Hence there must be an active arrow pointing into $\widehat{C}_h^r$. The starting point of this active arrow is a vertex $x_{i,S}^{mr+h}$ for some i and good subset S of P_i^{mr+h}. The set S corresponds to a group assignment of F_i that satisfies C_h. Since the arrow is active $x_{i,S}^{mr+h} \in Z'$, and by the discussion above we have that $P_i^{mr+h} \cap Z' = S$. Now, $S = P_i^{mr+h} \cap Z'$ and S is equal to $P_i^{mr+1} \cap Z'$ and hence the assignment to the variables of F_i satisfies C_h. Since this holds for all clauses, this concludes the proof. □

Lemma 2.25. $\mathrm{pw}(G) \leq t(p+1) + 10p3^p$.

Proof. We show how to search the graph using at most $t(p+1) + 10p3^p$ searchers. The strategy consists of $m(tp+1)$ rounds numbered from round 0 to round $m(tp+1) - 1$. Each round has t

stages, numbered from 1 to t. In the beginning of round k there is a searcher on $p_{i,j}^{3k+1}$ and ρ_i^k for every $1 \leq i \leq t$, $1 \leq j \leq p$. Let r and $1 \leq h \leq m$ be integers such that $k+1 = rm + h$.Recall, that as we go around $\widehat{C}_h^r$ counterclockwise we first encounter vertices corresponding to group assignments of F_1, then to assignments of F_2 and so on. In the beginning of round k we place a searcher on the first vertex on $\widehat{C}_h^r$ that corresponds to an assignment of F_1. If $\widehat{C}_h^r$ contains a dummy vertex, we place a searcher on this vertex as well. These two searchers will remain on their respective vertices throughout the round. In the beginning of stage s of round k we will assume that the vertices on the cycle $\widehat{C}_h^r$ corresponding to group assignments of $F_{s'}$, $s' < s$ have already been cleaned, and in the beginning of every stage $s > 1$, there is a searcher standing on the first vertex corresponding to a group assignment of F_s.

In stage s of round k, we place searchers on all vertices of P_s^k, A_s^k, B_s^k, Q_s^k, L_s^k, R_s^k, Y_s^k and all vertices of cycles $X_{s,S}^k$ for every good subset S of P_s^k, on all vertices of arrows starting or ending in such cycles, and on all vertices of $\widehat{C}_h^r$ corresponding to group assignments of F_s. In total this amounts to less than $10p3^p$ vertices.

In the last part of stage s of round k, we place searchers on $p_{s,j}^{3(k+1)+1}$ for every $1 \leq j \leq p$ and on ρ_s^{k+1}. Then we remove all the searchers that were placed out in the first part of phase s except for the searcher on the last vertex on $\widehat{C}_h^r$ corresponding to a group assignment of F_s. Unless $s = 1$ there is also a searcher on the last vertex on $\widehat{C}_h^r$ corresponding to a group assignment of F_{s-1}. We remove this searcher, and the next stage can commence. In the end of the last stage of round k we remove all the searchers from $\widehat{C}_h^r$. Then the last stage can commence. At any point in time, at most $t(p+1) + 10p3^p$ searchers are placed on G. □

Proof (of Theorem 2.22). Suppose Odd Cycle Transversal can be solved in time $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ for some $\epsilon > 0$. Then there is an $\epsilon' < 1$ such that $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)} \leq 3^{\epsilon' \mathrm{pw}(G)} \cdot n^{O(1)}$. We choose p large enough such that $\epsilon' \cdot \frac{p+1}{p-1} = \delta' < 1$. Given an instance of SAT we construct an instance of Odd Cycle Transversal using the above construction and the chosen value of p. Then we solve the Odd Cycle Transversal instance using the $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ time algorithm. Correctness is ensured by Lemmata 2.23 and 2.24. Lemma 2.25 yields that the total time taken is upper bounded by $(3-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)} \leq 3^{\epsilon' \mathrm{pw}(G)} \cdot n^{O(1)} \leq 3^{\epsilon'(t(p+1)+f(\epsilon'))} \cdot n^{O(1)} \leq 3^{\epsilon' \lceil \frac{n}{\lfloor p \log 3 \rfloor} \rceil (p+1)} \cdot n^{O(1)} \leq 3^{\epsilon' \frac{n(p+1)}{\lfloor p \log 3 \rfloor}} \cdot n^{O(1)} \leq 3^{\epsilon' \frac{n(p+1)}{(p-1)\log 3}} \cdot n^{O(1)} \leq 3^{\delta' \frac{n}{\log 3}} \cdot n^{O(1)} \leq 2^{\delta' n} \cdot n^{O(1)} = (2-\delta)^n \cdot n^{O(1)}$ for some $\delta > 01$. □

2.7 Partition Into Triangles

A *triangle packing* in a graph G is a collection of pairwise disjoint vertex sets $S_1, S_2, \ldots S_t$ in G such that S_i induces a triangle in G for every i. The size of the packing is t. If $V(G) = \bigcup_{i \leq t} S_i$, then the collection $S_1 \ldots S_t$ is a *partition of G into triangles*. In the Triangle Packing problem, we are given a graph G and an integer t and asked whether there is a triangle packing in G of size at least t. In the Partition Into Triangles problem, we are given a graph G and asked whether G can be partitioned into triangles. Notice that since Partition Into Triangles is the special case of Triangle Packing when the number of triangles is the number of vertices divided by 3, the bound of Theorem 2.26 holds for Triangle Packing as well.

Theorem 2.26. *If* Partition Into Triangles *can be solved in time $(2-\epsilon)^{\mathrm{pw}(G)} \cdot n^{O(1)}$ for $\epsilon > 0$, then* SAT *can be solved in $(2-\delta)^n \cdot n^{O(1)}$ time for some $\delta > 0$.*

Construction. first show the lower bound for Triangle Packing and then modify our construction to also work for the more restricted Partition Into Triangles problem. Given an

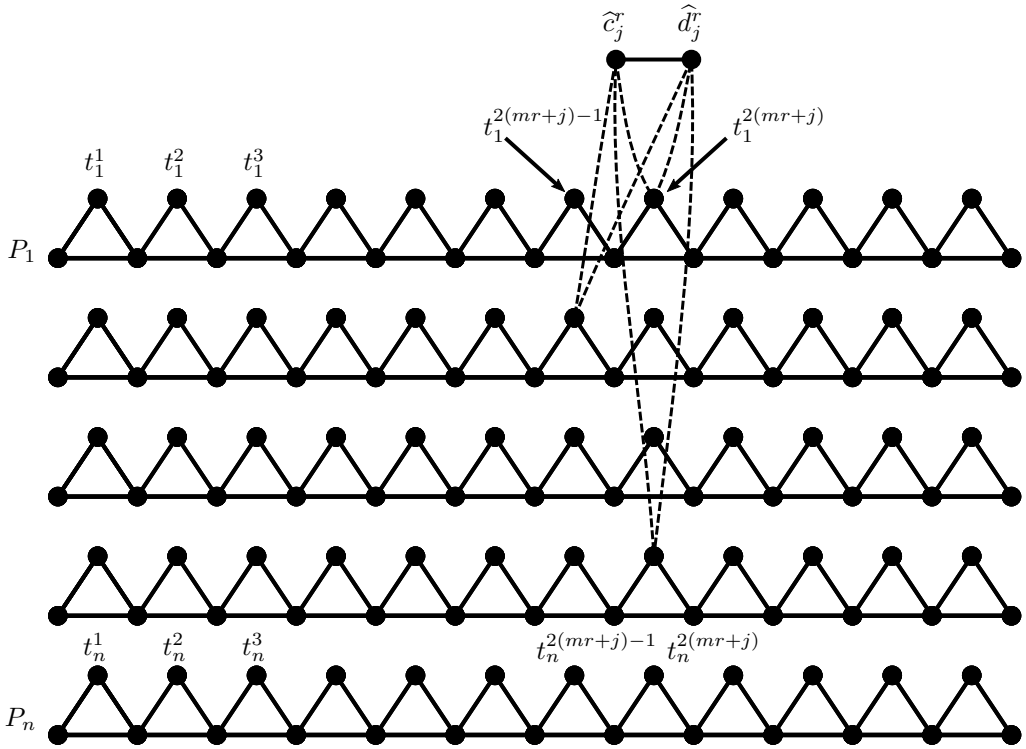

Figure 2.9: Reduction to Triangle Packing, showing how the vertices $\widehat{c}_j^r$ and $\widehat{d}_j^r$ representing clause $C_j = (x_1 \vee \bar{x}_2 \vee x_4)$ are connected to the paths $P_1, \ldots, P_n$.

instance ϕ of SAT we construct a graph G as follows (see Figure 2.9). For every variable v_i, we make a path P_i on $2m(n+1)+1$ vertices. We denote the ℓ-th vertex of P_i by p_i^ℓ. For every i, we add a set T_i of $2m(n+1)$ vertices, and let the ℓ-th vertex of T_i be denoted t_i^ℓ. For every $1 \leq \ell \leq 2m(n+1)$ we add the edges $t_i^\ell p_i^\ell$ and $t_i^\ell p_i^{\ell+1}$.

For every clause C_j, we add $n+1$ gadgets corresponding to the clause. In particular, for every $0 \leq r \leq n$ we do the following. First we add the vertices $\widehat{c}_j^r$ and $\widehat{d}_j^r$ and the edge $\widehat{c}_j^r \widehat{d}_j^r$. For every variable v_i that occurs in C_j positively, we add the edges $\widehat{c}_j^r t_i^{2(mr+j)}$ and $\widehat{d}_j^r t_i^{2(mr+j)}$. For every variable v_i that occurs in C_j negated, we add the edges $\widehat{c}_j^r t_i^{2(mr+j)-1}$ and $\widehat{d}_j^r t_i^{2(mr+j)-1}$. Doing this for every r and every clause C_j concludes the construction of G.

Lemma 2.27. *If ϕ satisfiable, then G has a triangle packing of size $mn(n+1)+m(n+1)$.*

Proof. Consider a satisfying assignment to ϕ. For every variable v_i that is set to true and integer $1 \leq \ell \leq m(n+1)$, we add $\{t_i^{2l-1}, p_i^{2l-1}, p_i^{2l}\}$ to the triangle packing. For every variable v_i that is set to false and integer $1 \leq \ell \leq m(n+1)$, we add $\{t_i^{2l}, p_i^{2l}, p_i^{2l+1}\}$ to the triangle packing. For every clause C_j, there is a literal set to true. Suppose this literal corresponds to the variable v_i. Notice that if v_i occurs positively in C_j, then v_i is set to true, and if it occurs negatively it is set to false. For each $0 \leq r \leq n$, if v_i occurs positively in C_j, then $t_i^{2(mr+j)}$ has not yet been used in any triangle, so we can add $\{\widehat{c}_j^r, \widehat{d}_j^r, t_i^{2(mr+j)}\}$ to the triangle packing. On the other hand, if v_i occurs negated in C_j, then $t_i^{2(mr+j)-1}$ has not yet been used in any triangle, so we can add $\{\widehat{c}_j^r, \widehat{d}_j^r, t_i^{2(mr+j)-1}\}$ to the triangle packing. In total $mn(n+1)+m(n+1)$ triangles are packed. □

Lemma 2.28. *If G has a triangle packing of size $mn(n+1)+m(n+1)$, then ϕ satisfiable.*

Proof. Observe that for any j and r, every triangle that contains $\widehat{c}_j^r$ also contains $\widehat{d}_j^r$ and vice versa. Furthermore, if we remove all the vertices $\widehat{c}_j^r$ and $\widehat{d}_j^r$ for every j and r from G we obtain a disconnected graph with n connected components, $G[T_i \cup V(P_i)]$ for every i. Thus, the only way to pack $mn(n+1) + m(n+1)$ triangles in G is to pack $mn(n+1)$ triangles in each component $G[T_i \cup V(P_i)]$ and in addition make sure that every pair $(\widehat{c}_j^r, \widehat{d}_j^r)$ is used in some triangle in the packing.

The only way to pack $mn(n+1)$ triangles in a component $G[T_i \cup V(P_i)]$ is to use every second triangle of the form $\{t_i^\ell, p_i^\ell, p_i^{\ell+1}\}$, except possibly at one point where two triangles on this form are skipped. By the pigeon hole principle there is an $0 \leq r \leq n$ such that for every i, every second triangle of the form $\{t_i^{2mr+\ell}, p_i^{2mr+\ell}, p_i^{2mr+\ell+1}\}$ for $1 \leq \ell \leq 2m$ is used. We make an assignment to the variables of ϕ as follows. For every i such that $\{t_i^{2mr+1}, p_i^{2mr+1}, p_i^{2mr+2}\}$ is used, v_i is set to true, and otherwise $\{t_i^{2mr+2}, p_i^{2mr+2}, p_i^{2mr+3}\}$ is used in the packing and v_i is set to false. We prove that this assignment satisfies ϕ.

For every j, the pair $(\widehat{c}_j^r, \widehat{d}_j^r)$ is used in some triangle in the packing. This triangle either contains $t_i^{2(mr+j)}$ or $t_i^{2(mr+j)-1}$ for some i. If it contains $t_i^{2(mr+j)}$, then v_i occurs positively in C_j. Furthermore, since the triangle packing contains every second triangle of the form $\{t_i^{2mr+\ell}, p_i^{2mr+\ell}, p_i^{2mr+\ell+1}\}$ for $1 \leq \ell \leq 2m$, it follows that the triangle packing contains $\{t_i^{2mr+1}, p_i^{2mr+1}, p_i^{2mr+2}\}$ and hence v_i is set to true. By an identical argument, if the triangle containing the pair $(\widehat{c}_j^r, \widehat{d}_j^r)$ contains $t_i^{2(mr+j)-1}$, then v_i occurs negated in C_j and v_i is set to false. This concludes the proof. □

We now modify the construction to work for Partition Into Triangles instead of Triangle Packing. Given the graph G as constructed from ϕ, we construct a graph G' as follows. For every $1 \leq i \leq n$ and $1 \leq l \leq m(n+1)$, we make a clique Q_i^ℓ on four vertices. The vertices of Q_i^ℓ are all adjacent to t_i^{2l} and to t_i^{2l-1}. For every $i < n$ and and $1 \leq l \leq m(n+1)$ we make all vertices of Q_i^ℓ adjacent to all vertices of Q_{i+1}^ℓ. Suppose that $2n+2$ is p modulo 3 for some $p \in \{0,1,2\}$. We remove p vertices from Q_n^ℓ for every $l \leq m(n+1)$.

Lemma 2.29. *G has a triangle packing of size α if and only if G' can be partitioned into triangles. Here, α is a non-negative integer.*

Proof. In the forward direction, consider a triangle packing of size α in G as constructed in Lemma 2.27. We can assume that the triangle packing has this form, because by Lemma 2.28 we have that ϕ is satisfiable.

For every fixed $1 \leq l \leq m(n+1)$, we proceed as follows. We know that there exists an i such that both t_i^{2l} and t_i^{2l-1} are used in the packing. For every $i' \neq i$, exactly one out of $t_{i'}^{2l}$ and $t_{i'}^{2l-1}$ is used in the packing. For each such i', we make a triangle containing the unused vertex out of $t_{i'}^{2l}$ and $t_{i'}^{2l-1}$ and two vertices of $Q_{i'}^\ell$. Then we "clean up" $Q_1^\ell, \ldots, Q_n^\ell$ as follows.

In particular, we start with the yet unused vertices of Q_1^ℓ. There are two of them. Make a triangle containing these two vertices and one vertex of Q_2^ℓ. Now Q_2^ℓ has one unused vertex left. Make a triangle containing this vertex and the two unused vertices of Q_3^ℓ. Continue in this fashion until arrive at $Q_{i'}^\ell$. At this point we have used 0, 1 or 2 vertices of $Q_{i'}^\ell$ a triangle containing some vertices in $Q_{i'-1}^\ell$. The case when we have used 0 vertices of $Q_{i'}^\ell$ also covers the case that $i' = 1$. If we only used 0 or 1 vertices of $Q_{i'}^\ell$, then we add a triangle that contains 3 vertices of $Q_{i'}^\ell$. If there are still unused vertices in $Q_{i'}^\ell$, then their number is either 1 or 2. We make a triangle containing these vertices and 1 or 2 of the unused vertices of $Q_{i'+1}^\ell$. Now we proceed to $Q_{i'+1}^\ell$ and continue in this manner until we reach Q_n^ℓ. Since the total number of vertices in $\bigcup_{j \leq n} Q_j^\ell$ is $4n - p$, we know that $2n-2$ of these vertices are used for triangles with vertices of G, and $2n + 2 - p$ is divisible by 3 the process described above will partition all the unused vertices of $\bigcup_{j \leq n} Q_j^\ell$ into triangles.

In the reverse direction, we argue that in any partitioning of G' into triangles, exactly α triangles must lie entirely within G. In fact, we argue that for any $l \leq m(n+1)$ exactly $n-1$ vertices out of $\bigcup_{i\leq n}\{t_i^{2l}, t_i^{2l-1}\}$ are used in triangles containing vertices from $\bigcup_{i\leq n} Q_i^\ell$.

Pick $1 \leq j \leq m$ and r such that $l = mr + j$. Exactly one out of $\bigcup_{i\leq n}\{t_i^{2l}, t_i^{2l-1}\}$ is in a triangle with $\widehat{c}_j^r$ and $\widehat{d}_j^r$. Furthermore, for each $i \leq n$ the vertex p_i^{2l} must be in a triangle either containing t_i^{2l} or t_i^{2l}. Hence, at most $n-1$ vertices out of $\bigcup_{i\leq n}\{t_i^{2l}, t_i^{2l-1}\}$ are used in triangles containing vertices from $\bigcup_{i\leq n} Q_i^\ell$. Furthermore, any triangle containing t_i^{2l} or $t_i^{2l-1}\}$ must either contain p_i^{2l}, $\widehat{c}_j^r$ or some vertex in $\bigcup_{i\leq n} Q_i^\ell$. Hence exactly $n-1$ vertices out of $\bigcup_{i\leq n}\{t_i^{2l}, t_i^{2l-1}\}$ are used in triangles containing vertices from $\bigcup_{i\leq n} Q_i^\ell$. Thus in the packing, exactly 3α vertices in G' are contained in triangles completely inside G, and hence G has a triangle packing of size α. □

To complete the proof for Partition Into Triangles we need to bound the pathwidth of G'.

Lemma 2.30. $\mathrm{pw}(G') \leq n + 10$.

Proof. We give a search strategy for G' that uses $n+10$ searchers. The strategy consists of $m(n+1)$ rounds and each round has n stages. In the beginning of round l, $1 \leq l \leq m(n+1)$, there are n searchers placed, one on each vertex p_i^{2l-1} for every i. Let r and $1 \leq j \leq m$ be integers such that $l = mr + j$. We place one searcher on $\widehat{c}_j^r$ and one on $\widehat{d}_j^r$. These two searchers will stay put throughout the duration of this round. In stage i of round l we place searchers on all vertices of Q_i^ℓ and Q_{i+1}^ℓ. Then we place searchers on t_i^{2l-1}, t_i^{2l}, p_i^{2l} and p_i^{2l+1}. At the end of stage i we remove the searchers from Q_i^ℓ, t_i^{2l-1}, t_i^{2l} and p_i^{2l}. We then proceed to the next stage. At the end of the round we remove the searchers from $\widehat{c}_j^r$ and $\widehat{d}_j^r$. Notice that now, there are searchers on p_i^{2l+1} for every i, and the next round can commence. □

Lemmata 2.27,2.28,2.29 and 2.30 prove Theorem 2.26.

CHAPTER 3

Treewidth and tight bounds on the complexity of Constraint Satisfaction Problems

Due to its generality, solving constraint satisfaction problems is NP-hard if we do not impose any additional restrictions on the possible instances. The main question that we want to explore in this chapter (and in many of the other chapters as well) is to understand what kind of structural restrictions on the primal graph of the instance can lead to improved algorithms. In this section, we consider only binary CSP instances, i.e., where every constraint involves at most two variables. The results presented here essentially go through for CSP instances with any fixed finite arity. The orginial publication where these results appeared [185] presented a generalization to the setting of the homomorphism problem for relational structures of a fixed signature, from which the results easily follow for any fixed finite arity. In order to avoid introducing another layer of formalisms (relational structures, homomorphisms, etc.), the results on the homomorphism problem are omitted from this dissertation.

Freuder [108] observed that if the treewidth of the primal graph is k, then CSP can be solved in time $n^{O(k)}$. (Here n is the size of the input; in the cases we are interested in in this chapter, the input size is polynomially bounded by the domain size and the number of variables.) The aim of this chapter is to investigate whether there exists any other structural property of the primal graph that can be exploited algorithmically to speed up the search for the solution.

Publications. The results presented in this chapter are based on a single-author publication that appeared in *Theory of Computing* [185] (an extended abstract appeared in the proceedings of the FOCS 2007 conference [180]).

Structural complexity of CSP instances. We would like to characterize the primal graphs that allow efficient solution of the CSP instance. But we have to be careful with the formalization of this question: if G is a graph with k vertices, then any CSP instance with primal graph G can be solved in time $n^{O(k)}$ by brute force. Therefore, restricting CSP to *any* fixed graph G makes it polynomial-time solvable. The real question is which *classes* of graphs make the problem polynomial-time solvable. Formally, for a class $\mathcal{G}$ of graphs, let CSP($\mathcal{G}$) be the class of all CSP instances where the primal graph of the instance is in $\mathcal{G}$. Note that this definition does not make any restriction on the constraint relations: it is possible that every constraint has a different constraint relation. If $\mathcal{G}$ has bounded treewidth, then CSP($\mathcal{G}$) is polynomial-time solvable. The converse is also true (under standard assumptions).

Theorem 3.1 (Grohe, Schwentick, Segoufin [127]; Grohe [123]). *If $\mathcal{G}$ is a recursively enumerable class of graphs, then CSP($\mathcal{G}$) is polynomial-time solvable if and only if $\mathcal{G}$ has bounded treewidth (assuming* FPT $\neq$ W[1]*).*

The results in [123, 127] are actually more general and are stated in terms of the conjunctive query and homomorphism problems, but it is easy to see that those results imply Theorem 3.1. The assumption FPT $\neq$ W[1] is a standard hypothesis of parameterized complexity (cf. [74, 87, 102]). Let us emphasize that the proof of Theorem 3.1 uses in an essential way the fact that the domain size can be arbitrarily large.

By Theorem 3.1, bounded treewidth is the only property of the primal graph that can make the problem polynomial-time solvable. However, Theorem 3.1 does not rule out the possibility that there is some structural property that may enable us to solve instances significantly faster than the treewidth-based algorithm of Freuder [108], that is, for some class $\mathcal{G}$ of graphs with unbounded treewidth, CSP($\mathcal{G}$) could be solved in time $n^{f(k)}$ where k is the treewidth and f is a slowly growing function such as $\sqrt{k}$ or $\log k$. The main result of this chapter is that this is not possible; the $n^{O(k)}$-time algorithm is essentially optimal for every class of graphs, up to an $O(\log k)$ factor in the exponent. Thus, in our specific setting, there is no other structural information beside treewidth that can be exploited algorithmically.

The formal statement of the main result of the chapter is the following (we denote by $\mathrm{tw}(G)$ the treewidth of G):

Theorem 3.2. *If there is a class $\mathcal{G}$ of graphs with unbounded treewidth, an algorithm $\mathbb{A}$, and a function f such that $\mathbb{A}$ correctly decides every binary CSP instance and the running time is $f(G)\|I\|^{o(\mathrm{tw}(G)/\log \mathrm{tw}(G))}$ for binary CSP($\mathcal{G}$) instances I with primal graph $G \in \mathcal{G}$, then ETH fails.*

Binary CSP($\mathcal{G}$) is the special case of CSP($\mathcal{G}$) where every constraint is binary, that is, involves two variables. Note that adding this restriction makes the statement of Theorem 3.2 stronger. Similarly, allowing the multiplicative factor $f(G)$ in the running time also makes the result stronger. We do not make any assumption on f, for example, we do not require that f be computable.

The main technical tool of the proof of Theorem 3.1 in [123, 127] is the Excluded Grid Theorem of Robertson and Seymour [208], which states that there is an unbounded function $g(k)$ such that every graph with treewidth at least k contains a $g(k) \times g(k)$ grid as minor. The basic idea of the proof in [123] is to show that CSP($\mathcal{G}$) is not polynomial-time solvable if $\mathcal{G}$ contains every grid and then this result is used to argue that CSP($\mathcal{G}$) is not polynomial for any $\mathcal{G}$ with unbounded treewidth, since in this case $\mathcal{G}$ contains every grid as minor. However, this approach does not work if we want a tighter lower bound, as in Theorem 3.2. The problem is that the function $g(k)$ is very slowly growing. For a long time, the function $g(k)$ was $o(\log k)$ in the best known proofs of the Excluded Grid Theorem [85, 208]. In a breakthrough result of Chekuri and Chuzhoy [55] and its subsequent improvements [64, 65], the dependence was improved to polynomial, i.e., $g(k) = \Omega(k^{1/c})$ for some integer $c \geq 1$ (around 19 in the latest proof). It is possible that this constant c is improved in the future, but lower bounds show that the function $g(k)$ cannot be better than $\Omega(k^{1/2})$ [209]. Therefore, if the only property of graphs with treewidth at least k that we use is that they have $g(k) \times g(k)$ grid minors, then we immediately lose a lot: as CSP on the $g(k) \times g(k)$ grid can be solved in time $\|I\|^{O(g(k))}$, no lower bound stronger than $\|I\|^{o(k^{1/2})}$ can be proved with this approach. Thus we need a characterization of treewidth that is tighter than the Excluded Grid Theorem.

The almost-tight bound of Theorem 3.2 is made possible by a new characterization of treewidth that is tight up to a logarithmic factor (Theorem 3.5). This result may be of independent interest. We generalize the notion of minors the following way. An *embedding* of H into G is a mapping ψ from $V(H)$ to connected subsets of G such that if $u, v \in V(H)$ are adjacent, then either $\psi(u) \cap \psi(v) \neq \emptyset$

or there is an edge connecting a vertex of $\psi(u)$ and a vertex of $\psi(v)$. The *depth* of the embedding is at most q if every vertex of G appears in the images of at most q vertices of H. Thus H has an embedding of depth 1 into G if and only if H is a minor of G.

We characterize treewidth by the "embedding power" of the graph in the following sense. If q is sufficiently large, then H has an embedding of depth q into G. For example, $q = |V(H)| \leq 2|E(H)|$ (assuming H has no isolated vertices) is certainly sufficient. However, we show that if the treewidth of G is at least k, then there is an embedding with depth $q = O(|E(H)| \log k/k)$, that is, the depth is a factor $O(k/\log k)$ better than in the trivial bound of $2|E(H)|$. We prove this result using the well-known characterization of treewidth by separators and an $O(\log k)$ integrality gap bound for the sparsest cut. The main idea of the proof of Theorem 3.2 is to use the embedding power of a graph with large treewidth to simulate a 3SAT instance efficiently.

Subgraph problems. Tight lower bounds on the exponent under ETH have previously been obtained in the framework of parameterized complexity. A basic result in this direction is due to Chen et al.:

Theorem 3.3 (Chen et al. [59, 60])**.** *There is no $f(k) \cdot n^{o(k)}$-time algorithm for k-Clique, unless ETH fails.*

Theorem 3.3 can be interpreted as a lower bound for the Subgraph Isomorphism problem (given two graphs G and H, decide if G is a subgraph of H). Using the color coding technique of Alon, Yuster, and Zwick [13], it is possible to solve Subgraph Isomorphism in time $f(|V(G)|) \cdot n^{O(\mathrm{tw}(G))}$. Theorem 3.3 and the fact that the treewidth of the k-clique is $k-1$ shows that it is not possible to improve the dependence on $\mathrm{tw}(G)$ in the exponent to $o(\mathrm{tw}(G))$, since in particular this would imply an $f(k) \cdot n^{o(k)}$-time algorithm for the k-Clique problem. However, this observation does not rule out the possibility that there is a special class of graphs (say, bounded degree graphs or planar graphs) where it is possible to improve the exponent to $o(\mathrm{tw}(G))$. In Section 3.4, we discuss lower bounds for Subgraph Isomorphism (more precisely, to its colored version) that follow from our CSP results.

Another important aspect of Theorem 3.3 is that it can be used to obtain lower bounds for other parameterized problems. W[1]-hardness proofs are typically done by parameterized reductions from k-Clique. It is easy to observe that a parameterized reduction implies a lower bound similar to Theorem 3.3 for the target problem, with the exact form of the lower bound depending on the way the reduction changes the parameter. Many of the more involved reductions use edge selection gadgets (see e.g., [98]). As the k-clique has $\Theta(k^2)$ edges, this means that the reduction increases the parameter to $\Theta(k^2)$ and we can conclude that there is no $f(k) \cdot n^{o(\sqrt{k})}$-time algorithm for the target problem (unless ETH fails). If we want to obtain stronger bounds on the exponent, then we have to avoid the quadratic blow-up of the parameter and do the reduction from a different problem. One possibility is to reduce from Subgraph Isomorphism, parameterized by the number of edges. Technically, it is usually more convenient to reduce from the Partitioned Subgraph Isomorphism problem, where the vertex set of the host graph H is partitiond into $|V(G)|$ classes, and we are looking for a subgraph mapping where the i-th vertex of G is mapped to the i-th class of H. In a reduction from Partitioned Subgraph Isomorphism, we need $|E(G)|$ edge selection gadgets, which usually implies that the new parameter is $\Theta(|E(G)|)$. Therefore, such a reduction and the following corollary obtained in Section 3.4 allows us to conclude that there is no $f(k) \cdot n^{o(k/\log k)}$-time algorithm for the target problem:

Corollary 3.4. *If* Partitioned Subgraph Isomorphism *can be solved in time $f(k)n^{o(k/\log k)}$, where f is an arbitrary function and $k = |E(G)|$ is the number of* edges *of the smaller graph G, then ETH fails.*

The use of Corollary 3.4 has become a standard technique when proving amost-tight lower bounds ruling out $f(k)n^{o(k/\log k)}$ time algorithms [35–37, 42, 68, 71, 92, 94, 128, 155, 158, 177, 190, 203]). It seems that edge representation of some sorts is required for many W[1]-hardness proofs, and for such problems basing the reduction on Theorem 3.3 would be able to rule out only algorithms with running time $f(k) \cdot n^{o(\sqrt{k})}$. For these problems, Corllary 3.4 is the only know way of obtaining a lower bound (almost) matching the $n^{O(k)}$ uppers bounds.

3.1 Preliminaries

We denote by $V(G)$ and $E(G)$ the set of vertices and the set of edges of the graph G, respectively. Given a graph G, the *line graph* $L(G)$ has one vertex for each edge of G, and two vertices of $L(G)$ are connected if and only if the corresponding edges in G share an endpoint. The line graph $L(K_k)$ of the complete graph K_k will appear repeatedly in the paper. Usually we denote the vertices of $L(K_k)$ by $v_{\{i,j\}}$ $(1 \leq i < j \leq k)$, where $v_{\{i_1,j_1\}}$ and $v_{\{i_2,j_2\}}$ are adjacent if and only if $\{i_1, j_1\} \cap \{i_2, j_2\} \neq \emptyset$.

A graph H is a *minor* of G if H can be obtained from G by a sequence of vertex deletions, edge deletions, and edge contractions. The following alternative definition will be more relevant to our purposes. An *embedding* of H into G is a mapping ψ from $V(H)$ to connected subsets of G such that if $u, v \in V(H)$ are adjacent, then either $\psi(u) \cap \psi(v) \neq \emptyset$ or there is an edge connecting a vertex of $\psi(u)$ and a vertex of $\psi(v)$. The *depth* of a vertex v of G is the size of the set $\{u \in V(H) \mid v \in \psi(u)\}$ and the depth of the embedding is the maximum of the depths of the vertices. It is easy to see that H is a minor of G if and only if H has an embedding of depth 1 into G, that is, the images are disjoint. To emphasize this connection, we will say that an embedding of depth 1 is a *minor mapping.*

In an equivalent way, we can use minors to define embeddings of a certain depth. Given a graph G and an integer q, we denote by $G^{(q)}$ the graph obtained by replacing every vertex with a clique of size q and replacing every edge with a complete bipartite graph on $q + q$ vertices. It is easy to see that H has an embedding of depth q into G if and only if H is a minor of $G^{(q)}$. The mapping ϕ that maps each vertex of G to the corresponding clique of $G^{(q)}$ will be called the *blow-up* mapping from G to $G^{(q)}$.

3.2 Embedding in a graph with large treewidth

If H is a graph with n vertices, then obviously H has an embedding of depth n into any (nonempty) G. If G has a clique of size k, then there is an embedding with depth at most n/k. Furthermore, even if G does not have a k-clique subgraph, but it does have a k-clique minor, then there is such an embedding with depth at most n/k. Thus a k-clique minor increases the "embedding power" of a graph by a factor of k. The main result of the section is that large treewidth implies a similar increase in embedding power. The following lemma states this formally:

Theorem 3.5. *There are computable functions $f_1(G)$, $f_2(G)$, and a universal constant c such that for every $k \geq 1$, if G is a graph with $\mathrm{tw}(G) \geq k$ and H is a graph with $|E(H)| = m \geq f_1(G)$ and no isolated vertices, then H has an embedding into G with depth at most $\lceil cm \log k/k \rceil$. Furthermore, such an embedding can be found in time $f_2(G)m^{O(1)}$.*

Using the equivalent characterization by minors, the conclusion of Theorem 3.5 means that H is a minor of $G^{(q)}$ for $q = \lceil cm \log k/k \rceil$. In the rest of the paper, we mostly use this notation.

The value $cm \log k/k$ is optimal up to an $O(\log k)$ factor, that is, it cannot be improved to $o(m/k)$. To see this, observe first that $\mathrm{tw}(G^{(q)}) = \Theta(q \cdot \mathrm{tw}(G))$ (cf. [125]). We use the fact that

the treewidth of a graph H with m edges can be $\Omega(m)$ (for example, bounded-degree expanders). Therefore, if $\mathrm{tw}(G) = k$, then the treewidth of $G^{(q)}$ for $q = o(m/k)$ is $o(m)$, making it impossible that H is a minor of $G^{(q)}$. Furthermore, Theorem 3.5 does not remain true if m is the number of vertices of H (instead of the number of edges). Let H be a clique on m vertices, and let G be a bounded-degree graph on $O(k)$ vertices with treewidth k. It is easy to see that $G^{(q)}$ has $O(q^2k)$ edges, hence H can be a minor of $G^{(q)}$ only if $q^2k = \Omega(m^2)$, that is, $q = \Omega(m/\sqrt{k})$. Note that it makes no sense to state in this form an analog of Theorem 3.5 where m is the number of vertices of H: the worst case happens if H is an m-clique, and the theorem would become a statement about embedding cliques. The requirement $m \geq f_1(G)$ is a technical detail: some of the arguments in the embedding technique requires H to be large.

The graph $L(K_k)$, that is, the line graph of the complete graph plays a central role in the proof of Theorem 3.5. The proof consists of two parts. In the first part (Section 3.2.1), we show that if $\mathrm{tw}(G) \geq k$, then a blow-up of $L(K_k)$ is a minor of an appropriate blow-up of G. This part of the proof is based on the characterization of treewidth by balanced separators and uses a result of Feige et al. [97] on the linear programming formulation of separation problems. Similar ideas were used in [125]; some of the arguments are reproduced here for the convenience of the reader. In the second part (Section 3.2.2), we show that every graph is a minor of an appropriate blow-up of $L(K_k)$.

3.2.1 Embedding $L(K_k)$ in G

Given a nonempty set W of vertices, we say that a set S of vertices is a *balanced separator* (with respect to W) if $|W \cap C| \leq |W|/2$ for every connected component C of $G \setminus S$. A k-separator is a separator S with $|S| \leq k$. The treewidth of a graph is closely connected with the existence of balanced separators:

Lemma 3.6 ([205], [102, Section 11.2])**.**

(a) If the graph G has treewidth greater than $3k$, then there is a set $W \subseteq V(G)$ of size $2k+1$ having no balanced k-separator.

(b) If the graph G has treewidth at most k, then every $W \subseteq V(G)$ has a balanced $(k+1)$-separator.

A *separation* is a partition of the vertices into three classes (A, B, S) ($S \neq \emptyset$) such that there is no edge between A and B. Note that it is possible that $A = \emptyset$ or $B = \emptyset$. The *sparsity* of the separation (A, B, S) (with respect to W) is defined in [97] as

$$\alpha^W(A, B, S) = \frac{|S|}{|(A \cup S) \cap W| \cdot |(B \cup S) \cap W|}. \tag{3.1}$$

We denote by $\alpha^W(G)$ the minimum of $\alpha^W(A, B, S)$ taken over every separation (A, B, S). It is easy to see that for every G and nonempty W, $1/|W|^2 \leq \alpha^W(G) \leq 1/|W|$ (the second inequality follows from the fact that the separation $(V(G) \setminus W, \emptyset, W)$ has sparsity exactly $1/|W|$). For our applications, we need a set W such that $\alpha^W(G)$ is close to the maximum possible, that is, $\Omega(1/|W|)$. The following lemma shows that the non-existence of a balanced separator can guarantee the existence of such a set W. The connection between balanced separators and sparse separations is well known, see for example [97, Section 6]. However, in our parameter setting a simpler argument is sufficient.

Lemma 3.7. *If $|W| = 2k+1$ and W has no balanced k-separator in a graph G, then $\alpha^W(G) \geq 1/(4k+1)$.*

Proof. Let (A, B, S) be a separation of sparsity $\alpha^W(G)$; we may assume that $|A \cap W| \geq |B \cap W|$, hence $|B \cap W| \leq k$. If $|S| > k$, then $\alpha^W(A, B, S) \geq (k+1)/(2k+1)^2 \geq 1/(4k+1)$. If $|S| \geq |(B \cup S) \cap W|$, then $\alpha^W(A, B, S) \geq 1/|(A \cup S) \cap W| \geq 1/(2k+1)$. Assume therefore that $|(B \cup S) \cap W| \geq |S| + 1$. Let S' be a set of $k - |S| \geq 0$ arbitrary vertices of $W \setminus (B \cup S)$. We claim that $S \cup S'$ is a balanced k-separator of W. Suppose that there is a component C of $G \setminus (S \cup S')$ that contains more than k vertices of W. Component C is either a subset of A or a subset of B. However, C cannot be a subset of B, since $|B \cap W| \leq k$. On the other hand, $|(A \setminus S') \cap W| \leq 2k + 1 - |(B \cup S) \cap W| - |S'| \leq 2k + 1 - (|S| + 1) - (k - |S|) \leq k$. □

Remark 3.8. Lemma 3.7 does not remain true in this form for larger W. For example, let K be a clique of size $3k + 1$, let us attach k degree-one vertices to a distinguished vertex x of K, and let us attach a degree-one vertex to every other vertex of K. Let W be the set of these $4k$ degree-one vertices. It is not difficult to see that W has no balanced k-separator. On the other hand, $S = \{x\}$ is a separator with sparsity $1/(k \cdot 3k)$, hence $\alpha^W(G) = O(1/k^2)$.

Let $W = \{w_1, \ldots, w_r\}$ be a set of vertices. A *concurrent vertex flow of value* ϵ is a collection of $|W|^2$ flows such that for every ordered pair $(u, v) \in W \times W$, there is a flow of value ϵ between u and v, and the total amount of flow going through each vertex is at most 1. A *flow* between u and v is a weighted collection of $u - v$ paths. A $u - v$ path contributes to the load of vertex u, of vertex v, and of every vertex between u and v on the path. In the degenerate case when $u = v$, vertex $u = v$ is the only vertex where the flow between u and v goes through, that is, the flow contributes to the load of only this vertex.

The maximum concurrent vertex flow can be expressed as a linear program the following way. For $u, v \in W$, let $\mathcal{P}_{uv}$ be the set of all $u - v$ paths in G, and for each $p \in \mathcal{P}_{uv}$, let variable $p^{uv} \geq 0$ denote the amount of flow that is sent from u to v along p. Consider the following linear program:

$$\begin{aligned}
&\text{maximize } \epsilon && && \\
&\text{s. t.} && && \\
&\sum_{p \in \mathcal{P}_{uv}} p^{uv} \geq \epsilon && \forall u, v \in W && \\
&\sum_{(u,v) \in W \times W} \sum_{p \in \mathcal{P}_{uv} : w \in p} p^{uv} \leq 1 && \forall w \in V && \text{(LP1)} \\
&p^{uv} \geq 0 && \forall u, v \in W, p \in \mathcal{P}_{uv} &&
\end{aligned}$$

The dual of this linear program can be written with variables $\{\ell_{uv}\}_{u,v \in W}$ and $\{s_v\}_{v \in V}$ the following way:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{v \in V} s_v & & & \\
\text{s. t.} \quad & & & & \\
& \sum_{w \in p} s_w \geq \ell_{uv} & & \forall u, v \in W, p \in \mathcal{P}_{uv} \ (*) & \\
& \sum_{(u,v) \in W \times W} \ell_{uv} \geq 1 & & (**) & \text{(LP2)} \\
& \ell_{uv} \geq 0 & & \forall u, v \in W & \\
& s_w \geq 0 & & \forall w \in V &
\end{aligned}
$$

We show that, in some sense, (LP2) is the linear programming relaxation of finding a separator with minimum sparsity. If there is a separation (A, B, S) with sparsity $\alpha^W(A, B, S)$, then (LP2) has a solution with value at most $\alpha^W(A, B, S)$. Set $s_v = \alpha^W(A, B, S)/|S|$ if $v \in S$ and $s_v = 0$ otherwise; the value of such a solution is clearly $\alpha^W(A, B, S)$. For every $u, v \in W$, set $\ell_{uv} = \min_{p \in \mathcal{P}_{uv}} \sum_{w \in p} s_w$ to ensure that inequalities (*) hold. To see that (**) holds, notice first that $\ell_{uv} \geq \alpha^W(A, B, S)/|S|$ if $u \in A \cup S$, $v \in B \cup S$, as every $u - v$ path has to go through at least one vertex of S. Furthermore, if $u, v \in S$ and $u \neq v$, then $\ell_{uv} \geq 2\alpha^W(A, B, S)/|S|$ since in this case a $u - v$ paths meets S in at least two vertices. The expression $|(A \cup S) \cap W| \cdot |(B \cup S) \cap W|$ counts the number of ordered pairs (u, v) satisfying $u \in (A \cup S) \cap W$ and $v \in (B \cup S) \cap W$, such that pairs with $u, v \in S \cap W$, $u \neq v$ are counted twice. Therefore,

$$\sum_{(u,v) \in W \times W} \ell_{uv} \geq (|(A \cup S) \cap W| \cdot |(B \cup S) \cap W|) \cdot \frac{\alpha^W(A, B, S)}{|S|} = 1,$$

which means that inequality (**) is satisfied.

The other direction is not true: a solution of (LP2) with value α does not imply that there is a separation with sparsity at most α. However, Feige et al. [97] proved that it is possible to find a separation whose sparsity is greater than that by at most a $O(\log |W|)$ factor (this result appears implicitly already in [172]):

Theorem 3.9 (Feige et al. [97], Leighton and Rao [172])**.** *If (LP2) has a solution with value α, then there is a separation with sparsity $O(\alpha \log |W|)$.*

We use (the contrapositive of) Theorem 3.9 to obtain a concurrent vertex flow in a graph with large treewidth. This concurrent vertex flow can be used to find an $L(K_k)$ minor in the blow-up of the graph in a natural way: the flow paths correspond to the edges of K_k.

Lemma 3.10. *Let G be a graph with $\mathrm{tw}(G) > 3k$. There are universal constants $c_1, c_2 > 0$ such that $L(K_k)^{(\lceil c_1 \log n \rceil)}$ is a minor of $G^{(\lceil c_2 \log n \cdot k \log k \rceil)}$, where n is the number of vertices of G.*

Proof. Since G has treewidth greater than $3k$, by Lemma 3.6(a), there is a subset W_0 of size $2k + 1$ that has no balanced k-separator. By Lemma 3.7, $\alpha^{W_0}(G) \geq 1/(4k + 1) \geq 1/(5k)$. Therefore, Theorem 3.9 implies that the dual linear program (LP2) has no solution with value less than $1/(c_0 5k \log(2k + 1))$, where c_0 is the constant hidden by the big O notation in Theorem 3.9. By linear programming duality, there is a concurrent flow of value at least $\alpha := 1/(c_0 5k \log(2k + 1))$ connecting the vertices of W_0; let p^{uv} be a corresponding solution of (LP1).

Let $W \subseteq W_0$ be a subset of k vertices. For each pair of vertices $(u, v) \in W \times W$, let us randomly and independently choose $\lceil \ln n \rceil$ paths $P_{u,v,1}, \ldots, P_{u,v,\lceil \ln n \rceil}$ of $\mathcal{P}_{uv}$ (here ln denotes the natural logarithm of n), where path p is chosen with probability

$$\frac{p^{uv}}{\sum_{p' \in \mathcal{P}_{uv}} (p')^{uv}} \leq \frac{p^{uv}}{\alpha}. \tag{3.2}$$

That is, we scale the values p^{uv} to obtain a probability distribution. Inequality (3.2) is true because the values p^{uv} satisfy (LP1). The expected number of times a path $p \in \mathcal{P}_{uv}$ is selected is $\lceil \ln n \rceil \cdot (p^{uv} / \sum_{p' \in \mathcal{P}_{uv}} (p')^{uv}) \leq \lceil \ln n \rceil \cdot p^{uv}/\alpha$. Thus the expected number of paths selected from $\mathcal{P}_{uv}$ that go through a vertex w is at most $\lceil \ln n \rceil \cdot \sum_{p \in \mathcal{P}_{uv} : w \in p} p^{uv}/\alpha$. Considering that we select $\lceil \ln n \rceil$ paths for every pair $(u, v) \in W \times W$, the expected number μ_w of selected paths containing w is at most $\lceil \ln n \rceil \cdot \sum_{(u,v) \in W \times W} \sum_{p \in \mathcal{P}_{uv} : w \in p} p^{uv}/\alpha$, which is at most $\lceil \ln n \rceil / \alpha$, since the values p^{uv} satisfy (LP1). We use the following standard Chernoff bound: for every $r > \mu_w$, the probability that more than $\mu_w + r$ of the $k^2 \ln n$ paths contain vertex w is at most $(\mu_w e/r)^r$. Thus the probability that more than $\mu_w + 10\lceil \ln n \rceil/\alpha \leq 11\lceil \ln n \rceil/\alpha$ of the paths contain w is at most $(\mu_w e/(10\lceil \ln n \rceil/\alpha))^{10\lceil \ln n \rceil/\alpha} \leq (1/e)^{10 \ln n} = 1/n^{10}$ (in the exponent, we used $\lceil \ln n \rceil/\alpha \geq \ln n$, since it can be assumed that $c_0 \geq 1$ and $\ln n \geq 1$). Therefore, with probability at least $1 - 1/n$, each vertex w is contained in at most $q := 11\lceil \ln n/\alpha \rceil$ paths. Note that $q \leq \lceil c_2 \log n \cdot k \log k \rceil$, for an appropriate value of c_2.

Let ϕ be the blow-up mapping from G to $G^{(q)}$. For each path $P_{u,v,i}$ in G, we define a path $P'_{u,v,i}$ in $G^{(q)}$. Let $P_{u,v,i} = p_1 p_2 \ldots p_r$. The path $P'_{u,v,i}$ we define consists of one vertex of $\phi(p_1)$, followed by one vertex of $\phi(p_2)$, ..., followed by one vertex of $\phi(p_r)$. The vertices are selected arbitrarily from these sets, the only restriction is that we do not select a vertex of $G^{(q)}$ that was already assigned to some other path $P'_{u',v',i'}$. Since each vertex w of G is contained in at most q paths, the q vertices of $\phi(w)$ are sufficient to satisfy all the paths going through w. Therefore, we can ensure that the $k^2\lceil \ln n \rceil$ paths $P'_{u,v,i}$ are pairwise disjoint in $G^{(q)}$.

The minor mapping from $L(K_k)^{(\lceil \ln n \rceil)}$ to $G^{(q)}$ is defined as follows. Let ψ be the blow-up mapping from $L(K_k)$ to $L(K_k)^{(\lceil \ln n \rceil)}$, and let $v_{\{1,2\}}, v_{\{1,3\}} \ldots, v_{\{k-1,k\}}$ be the $\binom{k}{2}$ vertices of $L(K_k)$, where $v_{\{i_1,i_2\}}$ and $v_{\{j_1,j_2\}}$ are adjacent if and only if $\{i_1, i_2\} \cap \{j_1, j_2\} \neq \emptyset$. Let $W = \{w_1, \ldots, w_k\}$. The $\lceil \ln n \rceil$ vertices of $\psi(v_{i,j})$ are mapped to the $\lceil \ln n \rceil$ paths $P'_{w_i,w_j,1}, \ldots, P'_{w_i,w_j,\lceil \ln n \rceil}$. Clearly, the images of the vertices are disjoint and connected. We have to show that this minor mapping maps adjacent vertices to adjacent sets. If $x \in \psi(v_{i_1,i_2})$ and $x' \in \psi(v_{j_1,j_2})$ are connected in $L(K_k)^{(\lceil \ln n \rceil)}$, then there is a $t \in \{i_1, i_2\} \cap \{j_1, j_2\}$. This means that the paths corresponding to x and x' both contain a vertex of the clique $\phi(w_t)$ in $G^{(q)}$, which implies that there is an edge connecting the two paths. □

With the help of the following proposition, we can make a small improvement on Lemma 3.10: the assumption $\mathrm{tw}(G) > 3k$ can be replaced by the assumption $\mathrm{tw}(G) \geq k$. This will make the result more convenient to use.

Proposition 3.11. *For every $k \geq 3$, $q \geq 1$, $L(K_{qk})$ is a subgraph of $L(K_k)^{(2q^2)}$.*

Proof. Let ϕ be a mapping from $\{1, \ldots, qk\}$ to $\{1, \ldots, k\}$ such that exactly q elements of $\{1, \ldots, qk\}$ are mapped to each element of $\{1, \ldots, k\}$. Let $v_{\{i_1,i_2\}}$ $(1 \leq i_1 < i_2 \leq qk)$ be the vertices of $L(K_{qk})$ and $u^t_{\{i_1,i_2\}}$ $(1 \leq i_1 < i_2 \leq k,\ 1 \leq t \leq 2q^2)$ be the vertices of $L(K_k)^{(2q^2)}$, with the usual convention that two vertices are adjacent if and only if the lower indices are not disjoint. Let $U_{\{i_1,i_2\}}$ be the clique $\{u^t_{\{i_1,i_2\}} \mid 1 \leq t \leq 2q^2\}$. Let us consider the vertices of $L(K_{qk})$ in some order. If $\phi(i_1) \neq \phi(i_2)$, then vertex $v_{\{i_1,i_2\}}$ is mapped to a vertex of $U_{\{\phi(i_1),\phi(i_2)\}}$ that was not already used for a previous vertex.

If $\phi(i_1) = \phi(i_2)$, then $v_{\{i_1,i_2\}}$ is mapped to a vertex $U_{\{\phi(i_1),\phi(i_1)+1\}}$ (where addition is modulo k). It is clear that if two vertices of $L(K_{qk})$ are adjacent, then the corresponding vertices of $L(K_k)^{(2q^2)}$ are adjacent as well. We have to verify that, for a given i_1, i_2, at most $2q^2$ vertices of $L(K_{qk})$ are mapped to the clique $U_{\{i_1,i_2\}}$. As $|\phi^{-1}(i_1)|$ and $|\phi^{-1}(i_2)|$ are both q, there are at most q^2 vertices $v_{\{j_1,j_2\}}$ with $\phi(j_1) = i_1$, $\phi(j_2) = i_2$. Furthermore, if $i_2 = i_1 + 1$, then there are $\binom{q}{2} \leq q^2$ additional vertices $v_{\{j_1,j_2\}}$ with $\phi(j_1) = \phi(j_2) = i_1$ that are also mapped to $U_{\{i_1,i_2\}}$. Thus at most $2q^2$ vertices are mapped to each clique $U_{\{i_1,i_2\}}$. □

Set $k' := 3k + 1 \leq 4k$. Using Prop. 3.11 with $q = 4$, we get that $L(K_{k'})^{(\lceil c_1 \log n \rceil/32)}$ is a subgraph of $L(K_k)^{(\lceil c_1 \log n \rceil)}$. Thus if $\mathrm{tw}(G) \geq k'$, then we can not only find a blowup of $L(K_k)$, but even a blowup of $L(K_{k'})$. By replacing k' with k, Lemma 3.10 can be improved the following way:

Lemma 3.12. *Let G be a graph with $\mathrm{tw}(G) \geq k$. There are universal constants $c_1, c_2 > 0$ such that $L(K_k)^{(\lceil c_1 \log n \rceil)}$ is a minor of $G^{(\lceil c_2 \log n \cdot k \log k \rceil)}$, where n is the number of vertices of G.* □

3.2.2 Embedding H in $L(K_k)$

As the second step of the proof of Theorem 3.5, we show that every (sufficiently large) graph H is a minor of $L(K_k)^{(q)}$ for $q = O(|E(H)|/k^2)$.

Lemma 3.13. *For every $k > 1$ there is a constant $n_k = O(k^4)$ such that for every G with $|E(G)| > n_k$ and no isolated vertices, the graph G is a minor of $L(K_k)^{(q)}$ for $q = \lceil 130|E(G)|/k^2 \rceil$. Furthermore, a minor mapping can be found in time polynomial in q and the size of G.*

Proof. We may assume that $k \geq 5$: otherwise the result is trivial, as $q \geq 2|E(G)| \geq |V(G)|$ and $L(K_k)^{(q)}$ contains a clique of size q. First we construct a graph G' of maximum degree 3 that contains G as a minor. This can be achieved by replacing every vertex v of G with a path on $d(v)$ vertices (where $d(v)$ is the degree of v in G); now we can ensure that the edges incident to v use distinct copies of v from the path. The new graph G' has exactly $2|E(G)|$ vertices.

We show that G', hence G, is a minor of $L(K_k)^{(q)}$. Take an arbitrary partition of $V(G')$ into $\binom{k}{2}$ classes $V_{\{i,j\}}$ $(1 \leq i < j \leq k)$ such that $|V_{\{i,j\}}| \leq \lceil |V|/\binom{k}{2} \rceil$ for every i, j. Let $v_{\{i,j\}}$ $(1 \leq i < j \leq k)$ be the vertices of $L(K_k)$, and let ϕ be the blow-up mapping from $L(K_k)$ to $L(K_k)^{(q)}$.

The minor mapping ψ from G' to $L(K_k)^{(q)}$ is defined the following way. First, if $u \in V_{\{i,j\}}$, then let $\psi(u)$ contain a vertex $\hat{u}$ from $\phi(v_{\{i,j\}})$. Observe that if edge e connects vertices $u_1 \in V_{\{i_1,j_1\}}$, $u_2 \in V_{\{i_2,j_2\}}$ and $\{i_1, j_1\} \cap \{i_2, j_2\} \neq \emptyset$ holds, then $\hat{u}_1$ and $\hat{u}_2$ are adjacent. In order to ψ be a minor mapping, we extends the sets $\psi(u)$ to ensure that the endpoints of e are mapped to adjacent sets even if $V_{\{i_1,j_1\}}$ and $V_{\{i_2,j_2\}}$ have disjoint indices.

Fix an arbitrary orientation of each edge of G'. For every quadruple (i_1, j_1, i_2, j_2) of distinct values with $i_1 < j_1$, $i_2 < j_2$, let E_{i_1,j_1,i_2,j_2} be the set of edges going from a vertex of $V_{\{i_1,j_1\}}$ to a vertex of $V_{\{i_2,j_2\}}$. Let us partition the set E_{i_1,j_1,i_2,j_2} into $k-4$ classes $E^{\ell}_{i_1,j_1,i_2,j_2}$ $(\ell \in \{1, \dots k\} \setminus \{i_1, j_1, i_2, j_2\})$ in an arbitrary way such that $|E^{\ell}_{i_1,j_1,i_2,j_2}| \leq \lceil |E_{i_1,j_1,i_2,j_2}|/(k-4) \rceil$. For each edge $\overrightarrow{uw} \in E^{\ell}_{i_1,j_1,i_2,j_2}$, we add a vertex of $\phi(v_{\{i_1,\ell\}})$ to $\psi(u)$ and a vertex of $\phi(v_{\{i_2,\ell\}})$ to $\psi(w)$; these two vertices are neighbors with each other and they are adjacent to $\hat{u}$ and $\hat{w}$, respectively. This ensures that $\psi(u)$ and $\psi(v)$ remain connected and there is an edge between $\psi(u)$ and $\psi(w)$. After repeating this step for every edge, ψ is clearly a minor mapping.

What remains to be shown is that the sets $\phi(v_{\{x,y\}})$ are large enough so that we can ensure that no vertex of $L(K_k)^{(q)}$ is assigned to more than one $\psi(u)$. Let us count how many vertices of $\phi(v_{\{x,y\}})$ are used when the minor mapping is constructed as described above. First, the image of each vertex u in $V_{\{x,y\}}$ uses one vertex $\hat{u}$ of $\phi(v_{\{x,y\}})$; together these vertices use at most $|V_{\{x,y\}}| \leq \lceil |V(G')|/\binom{k}{2} \rceil$

vertices from $\phi(v_{\{x,y\}})$. Furthermore, as described in the previous paragraph, for some quadruples (i_1, j_1, i_2, j_2) and integer ℓ, each edge of $E^{\ell}_{i_1,j_1,i_2,j_2}$ requires the use of an additional vertex from $\phi(v_{\{x,y\}})$. More precisely, this can happen only if $\ell = x$ and $y \in \{i_1, j_1, i_2, j_2\}$ or $\ell = y$ and $x \in \{i_1, j_1, i_2, j_2\}$. Thus the total number of vertices used from $\phi(v_{\{x,y\}})$ is at most

$$\begin{aligned}
&\lceil |V(G')|/\binom{k}{2}\rceil + \sum_{x\in\{i_1,j_1,i_2,j_2\}} |E^{y}_{i_1,j_1,i_2,j_2}| + \sum_{y\in\{i_1,j_1,i_2,j_2\}} |E^{x}_{i_1,j_1,i_2,j_2}| \\
&\quad \le |V(G')|/\binom{k}{2} + 1 + \sum_{x\in\{i_1,j_1,i_2,j_2\}} \lceil |E_{i_1,j_1,i_2,j_2}|/(k-4)\rceil + \sum_{y\in\{i_1,j_1,i_2,j_2\}} \lceil |E_{i_1,j_1,i_2,j_2}|/(k-4)\rceil \\
&\quad \le |V(G')|/\binom{k}{2} + \sum_{x\in\{i_1,j_1,i_2,j_2\}} |E_{i_1,j_1,i_2,j_2}|/(k-4) + \sum_{y\in\{i_1,j_1,i_2,j_2\}} |E_{i_1,j_1,i_2,j_2}|/(k-4) + 2k^4.
\end{aligned}$$

(The term $2k^4$ generously bounds the rounding errors, since it is greater than the number of terms in the sums.) The first sum counts only edges incident to some vertex of $V_{\{i,j\}}$ with $x \in \{i, j\}$ and each edge is counted at most once. Since each vertex has degree at most 3, the number of such edges is at most $3\sum_{x\in\{i,j\}} |V_{\{i,j\}}|$. Thus we can bound the first sum by $3(k-1)\lceil |V(G')|/\binom{k}{2}\rceil/(k-4) \le 12\lceil |V(G')|\binom{k}{2}\rceil$ (here we use $k \ge 5$). A similar argument applies for the second sum above, hence the number of vertices used from $\phi(v_{\{x,y\}})$ can be bounded as

$$\begin{aligned}
|V(G')|/\binom{k}{2} + 24\lceil |V(G')|/\binom{k}{2}\rceil + 2k^4 \le 25|V(G')|/\binom{k}{2} + 2k^4 + 24 \le 26|V(G')|/\binom{k}{2} \\
= 52|V(G')|/(k(k-1)) \le 65|V(G')|/k^2 = 130|E(G)|/k^2 \le q,
\end{aligned}$$

what we had to show (in the second inequality, we used that $|V(G')| = 2|E| \ge n_k$ is sufficiently large; in the third inequality, we used that $k \ge 5$ implies $k/(k-1) \le 5/4$). □

Putting together Lemma 3.12 and Lemma 3.13, we can prove the main result of the section:

Proof (of Theorem 3.5). Let $k := \mathrm{tw}(G)$, $n := |V(G)|$, and $f_1(G) := n_k + k^2\lceil c_1 \log n\rceil$, where n_k is the constant from Lemma 3.13 and c_1 is the constant from Lemma 3.12. Assume that $|E(H)| = m \ge f_1(G)$. By Lemma 3.13, H is a minor of $L(K_k)^{(q)}$ for $q := \lceil 130m/k^2\rceil$ and a minor mapping ψ_1 can be found in polynomial time. Let $q' := \lceil q/\lceil c_1 \log n\rceil\rceil$; clearly, H is a minor of $L(K_k)^{(q'\lceil c_1 \log n\rceil)}$. Observe that m is large enough such that $130m/k^2 \ge 1$ and $q/\lceil c_1 \log n\rceil \ge 1$ holds, hence $q' \le c' \cdot m/(k^2 \cdot \log n)$ for an appropriate constant c'.

By Lemma 3.12, $L(K_k)^{(\lceil c_1 \log n\rceil)}$ is a minor of $G^{(\lceil c_2 \log n \cdot k \log k\rceil)}$ and a minor mapping ψ_2 can be found in time $f_2(G)$ by brute force, for some function $f_2(G)$. Therefore, $L(K_k)^{(q'\lceil c_1 \log n\rceil)}$ is a minor of $G^{(q'\lceil c_2 \log n \cdot k \log k\rceil)}$ and it is straightforward to obtain the corresponding minor mapping ψ_3 from ψ_2. We may assume $c_2 \log n \cdot k \log k \ge 1$, otherwise the theorem automatically holds if we set c sufficiently large. Since $q'\lceil c_2 \log n \cdot k \log k\rceil \le c' \cdot m/(k^2 \cdot \log n) \cdot (2c_2 \log n \cdot k \log k) \le cm \log k/k$ for an appropriate constant c, we have that H is a minor of $G^{\lceil cm \log k/k\rceil}$. The corresponding minor mapping is the composition $\psi_3 \circ \psi_1$. Observe that each step can be done in polynomial time, except the application of Lemma 3.12, which takes $f_2(G)$ time. Thus the total running time can be bounded by $f_2(G)m^{O(1)}$. □

3.3 Complexity of binary CSP

In this section, we prove our main result for binary CSP (Theorem 3.2). The main strategy of the proof is the following. First we show that a 3SAT formula ϕ with m clauses can be turned into an equivalent binary CSP instance I of size $O(m)$ (Lemma 3.14). Here "equivalent" means that ϕ is satisfiable if and only if I has a solution. By the embedding result of Theorem 3.5, for every $G \in \mathcal{G}$, the primal graph of I is a minor of $G^{(q)}$ for an appropriate q. This implies that we can simulate I with a CSP instance I' whose primal graph is G (Lemma 3.15 and Lemma 3.16). Now we can use the assumed algorithm for CSP($\mathcal{G}$) to solve instance I', and thus decide the satisfiability of formula ϕ. If the treewidth of G is sufficiently large, then the assumed algorithm is much better than the treewidth-based algorithm. This translates into a $2^{o(m)}$ algorithm for the 3SAT instance, violating Hypothesis 1.2 and hence the ETH fails.

Lemma 3.14. *Given an instance of 3SAT with n variables and m clauses, it is possible to construct in polynomial time an equivalent CSP instance with $n + m$ variables, $3m$ binary constraints, and domain size 3.*

Proof. Let ϕ be a 3SAT formula with n variables and m clauses. We construct an instance of CSP as follows. The CSP instance contains a variable x_i ($1 \leq i \leq n$) corresponding to the i-th variable of ϕ and a variable y_j ($1 \leq j \leq m$) corresponding to the j-th clause of ϕ. Let $D = \{1, 2, 3\}$ be the domain. We try to describe a satisfying assignment of ϕ with these $n + m$ variables. The intended meaning of the variables is the following. If the value of variable x_i is 1 (or 2), then this represents that the i-th variable of ϕ is true (or false, respectively). If the value of variable y_j is ℓ, then this represents that the j-th clause of ϕ is satisfied by its ℓ-th literal. To ensure consistency, we add $3m$ constraints. Let $1 \leq j \leq m$ and $1 \leq \ell \leq 3$, and assume that the ℓ-th literal of the j-th clause is a positive occurrence of the i-th variable. In this case, we add the binary constraint $(x_i = 1 \vee y_j \neq \ell)$: either x_i is true or some other literal satisfies the clause. Similarly, if the ℓ-th literal of the j-th clause is a negated occurrence of the i-th variable, then we add the binary constraint $(x_i = 2 \vee y_j \neq \ell)$. It is easy to verify that if ϕ is satisfiable, then we can assign values to the variables of the CSP instance such that every constraint is satisfied, and conversely, if the CSP instance has a solution, then ϕ is satisfiable. □

If G_1 is a minor of G_2, then an instance with primal graph G_1 can be easily simulated by an instance with primal graph G_2: each variable of G_1 is simulated by a connected set of variables in G_2 that are forced to be equal.

Lemma 3.15. *Assume that G_1 is a minor of G_2. Given a binary CSP instance I_1 with primal graph G_1 and a minor mapping ψ from G_1 to G_2, it is possible to construct in polynomial time an equivalent instance I_2 with primal graph G_2 and the same domain.*

Proof. For simplicity, we assume that both G_1 and G_2 are connected; the proof can be easily extended to the general case. If G_2 is connected, then we may assume that ψ is onto. For each pair (x, y) such that xy is and edge of G_2, we add a constraint as follows. If $\psi^{-1}(x) = \psi^{-1}(y)$, then the new constraint is $\langle (x, y), \{(t, t) \mid t \in D\}\rangle$. If $\psi^{-1}(x) \neq \psi^{-1}(y)$ and there is a constraint $\langle (\psi^{-1}(x), \psi^{-1}(y)), R\rangle$, then the new constraint is $\langle (x, y), R\}\rangle$. Otherwise, the new constraint is $\langle (x, y), D \times D\}\rangle$. Clearly, the primal graph of I_2 is G_2.

Assume that I_1 has a solution $f_1 : V_1 \to D$. Then $f_2(v) = f_1(\psi^{-1}(v))$ is a solution of I_2. On the other hand, if I_2 has a solution $f_2 : V_2 \to D$, then we claim that $f_2(x) = f_2(y)$ holds if $\psi^{-1}(x) = \psi^{-1}(y)$. This follows from the way we defined the constraints of I_2 and from the fact that $\psi(x)$ is connected. Therefore, we can define $f_1 : V_1 \to D$ as $f_1(v) = f_2(v')$, where v' is an

arbitrary member of $\psi(v)$. To see that a constraint $c_i = \langle (u,v), R_i \rangle$ of I_1 is satisfied, observe that there is a constraint $\langle (u',v'), R_i \rangle$ in I_2 for some $u' \in \psi(u)$, $v' \in \psi(v)$. This means that $(f_1(u), f_1(v)) = (f_2(u'), f_2(v')) \in R_i$, hence the constraint is satisfied. □

An instance with primal graph $G^{(q)}$ can be simulated by an instance with primal graph G if we set the domain to be the q-tuples of the original domain.

Lemma 3.16. *Given a binary CSP instance $I_1 = (V_1, D_1, C_1)$ with primal graph $G^{(q)}$ (where G has no isolated vertices), it is possible to construct (in time polynomial in the size of the* output*) an equivalent instance $I_2 = (V_2, D_2, C_2)$ with primal graph G and $|D_2| = |D_1|^q$.*

Proof. Let ψ be the blow-up mapping from G to $G^{(q)}$ and let $D_2 = D_1^q$, that is, D_2 is the set of q-tuples of D_1. For every $v \in V_2$, there is a natural bijection between the elements of D_2 and the $|D_1|^q$ possible assignments $f : \psi(v) \to D_1$. For each edge v_1v_2 of G, we add a constraint $c_{v_1,v_2} = \langle (v_1, v_2), R_{v_1,v_2} \rangle$ to I_2 as follows. Let $(x_1, x_2) \in D_2 \times D_2$. For $i = 1, 2$, let g_i be the assignment of $\psi(v_i)$ corresponding to $x_i \in D_2$. The two assignments together define an assignment $g : \psi(v_1) \cup \psi(v_2) \to D$ on the union of their domains. Let $I[\psi(v_1) \cup \psi(v_2)]$ be the induced instance that has variables $\psi(v_1) \cup \psi(v_2)$ and contains only those constraint whose scope is contained in $\psi(v_1) \cup \psi(v_2)$. We define the relation R_{v_1,v_2} such that (x_1, x_2) is a member of R_{v_1,v_2} if and only if the assignment g corresponding to g_1, g_2 is a solution of $I[\psi(v_1) \cup \psi(v_2)]$.

Assume that I_1 has a solution $f_1 : V_1 \to D_1$. For every $v \in V_2$, let us define $f_2(v)$ to be the member of D_2 corresponding to the assignment f_1 restricted to $\psi(v)$. It is easy to see that f_2 is a solution of I_2: this follows from the trivial fact that for every edge v_1v_2 in G, assignment f_1 restricted to $\psi(v_1) \cup \psi(v_2)$ is a solution of $I_1[\psi(v_1) \cup \psi(v_2)]$.

Assume now that I_2 has a solution $f_2 : V_2 \to D_2$. For every $v \in V_2$, there is an assignment $f_v : \psi(v) \to D_1$ corresponding to $f_2(v)$. These assignments together define an assignment $f_1 : V_1 \to D_1$. We claim that f_1 is a solution of I_1. Let $c_{u,v} = \langle (u,v), R \rangle$ be an arbitrary constraint of I_1. Assume that $u \in \psi(u')$ and $v \in \psi(v')$. If $u' \neq v'$, then $u'v'$ is an edge of G, hence there is a corresponding constraint $c_{u',v'}$ in I_2. The way $c_{u',v'}$ is defined ensures that f_1 restricted to $\psi(u') \cup \psi(v')$ is a solution of $I_1[\psi(u') \cup \psi(v')]$. In particular, this means that $c_{u,v}$ is satisfied in f_1. If $u' = v'$, then there is an edge $u'w$ in G (since G has no isolated vertices), and the corresponding constraint $c_{u',w}$ ensures that f_1 satisfies $c_{u,v}$. □

Now we are ready to prove the main result:

Proof (of Theorem 3.2). Assume that there is an algorithm $\mathbb{A}$ that correctly decides every CSP instance and whose running time can be bounded by $f(G)\|I\|^{\mathrm{tw}(G)/(\log \mathrm{tw}(G) \cdot \iota(\mathrm{tw}(G)))}$ for instances with $G \in \mathcal{G}$, where ι is an unbounded function. We may assume that ι is nondecreasing and $\iota(1) \geq 1$. We present a reduction from 3SAT to CSP($\mathcal{G}$) such that this reduction, together with the assumed algorithm $\mathbb{A}$ for CSP($\mathcal{G}$), gives an algorithm $\mathbb{B}$ that is able to solve m-clause 3SAT in time $2^{o(m)}$. Lemma 3.14, Theorem 3.5, and Lemmas 3.15 and 3.16 show a way of solving a 3SAT instance by reducing it to a CSP instance having a particular primal graph G. A crucial point of the reduction is how to select an appropriate G from $\mathcal{G}$. The higher the treewidth of G, the more we gain in the running time. However, G has to be sufficiently small such that some additional factors (such as the time spent on finding G) are not too large.

Given an m-clause 3SAT formula ϕ and a graph $G \in \mathcal{G}$, algorithm $\mathbb{A}$ can be used to decide the satisfiability of ϕ in the following way. By Lemma 3.14, ϕ can be turned into a binary CSP instance I_1 with $O(m)$ constraints and domain size 3. Let H be the primal graph of I_1. For simplicity, we assume that G has no isolated vertices as they can be handled in a straightforward way. By

Theorem 3.5, H is a minor of $G^{(q)}$ for $q = O(m \log k/k)$ and we can find a minor mapping ψ in time $f_2(G)m^{O(1)}$. Therefore, by Lemma 3.15, I_1 can be turned into an instance I_2 with primal graph $G^{(q)}$, which, by Lemma 3.16, can be turned into an instance I_3 with primal graph G and domain size 3^q. Now we can use algorithm $\mathbb{A}$ to solve instance I_3.

We shall refer to this way of solving the 3SAT instance ϕ as "running algorithm $\mathbb{A}[\phi, G]$." Let us determine the running time of $\mathbb{A}[\phi, G]$. The two dominating terms are the time required to find the minor mapping from H to $G^{(q)}$ and the time required to run $\mathbb{A}$ on I_3. Note that $\|I_3\| = O(|E(G)|3^{2q})$: there are $|E(G)|$ constraints and each binary constraint contains at most $3^q \cdot 3^q$ pairs. Let k be the treewidth of G. The total running time of $\mathbb{A}[\phi, G]$ can be bounded by

$$\begin{aligned} f_2(G)m^{O(1)} + f(G)\|I_3\|^{k/(\log k \cdot \iota(k))} &= f_2(G)m^{O(1)} + f(G)|E(G)|^{k/(\log k \cdot \iota(k))} \cdot 3^{2qk/(\log k \cdot \iota(k))} \\ &= \hat{f}(G)m^{O(1)} \cdot 2^{O(qk/(\log k \cdot \iota(k)))} = \hat{f}(G)m^{O(1)} \cdot 2^{O(m/\iota(k))} \end{aligned}$$

for an appropriate function $\hat{f}(G)$.

Let us fix an arbitrary easy-to-compute enumeration $G_1, G_2, \ldots$ of all graphs. Given an m-clause 3SAT formula ϕ, we first spend m steps to enumerate graphs from $\mathcal{G}$; let G_ℓ (for some $\ell \leq m$) be the last graph enumerated (we assume that m is sufficiently large such that $\ell \geq 1$). Next we start simulating the algorithms $\mathbb{A}[\phi, G_1]$, $\mathbb{A}[\phi, G_2]$, $\ldots$, $\mathbb{A}[\phi, G_\ell]$ in *parallel*. When one of the simulations stops and returns an answer, then we stop all the simulations and return the answer. It is clear that this algorithm will correctly decide the satisfiability of ϕ.

We claim that there is a universal constant C such that for every s, there is an m_s such that for every $m > m_s$, the running time of $\mathbb{B}$ is $(m \cdot 2^{m/s})^C$ on an m-clause formula. Clearly, this means that the running time of $\mathbb{B}$ is $2^{o(m)}$.

Let k_s be the smallest positive integer such that $\iota(k_s) \geq s$ (as ι is unbounded, this is well defined). Let i_s be the smallest positive integer such that $G_{i_s} \in \mathcal{G}$ and $\mathrm{tw}(G_{i_s}) \geq k_s$ (as $\mathcal{G}$ has unbounded treewidth, this is also well defined). Set m_s sufficiently large such that $m_s \geq \hat{f}(G_{i_s})$ and the enumeration of all graphs reaches G_{i_s} in less then m_s steps. This means that if we run $\mathbb{B}$ on a 3SAT formula ϕ with $m \geq m_s$ clauses, then $\mathbb{A}[\phi, G_{i_s}]$ will be one of the ℓ simulations started by $\mathbb{B}$. The simulation of $\mathbb{A}[\phi, G_{i_s}]$ terminates in

$$\hat{f}(G_{i_s})m^{O(1)} \cdot 2^{O(m/\iota(\mathrm{tw}(G_{i_s})))} = m \cdot m^{O(1)} \cdot 2^{O(m/s)}$$

steps. Taking into account that we simulate $\ell \leq m$ algorithms in parallel and all the simulations are stopped not later than the termination of $\mathbb{A}[\phi, G_{i_s}]$, the running time of $\mathbb{B}$ can be bounded polynomially by the running time of $\mathbb{A}[\phi, G_{i_s}]$. Therefore, there is a constant C such that the running time of $\mathbb{B}$ is $(m \cdot 2^{m/s})^C$, as required. □

3.4 Complexity of subgraph problems

Subgraph Isomorphism is a basic graph-theoretic problem: given graphs G and H, we have to decide if G is a subgraph of H. That is, we have to find an injective mapping $\phi : V(G) \to V(H)$ such that if u and v are adjacent in the smaller graph G, then $\phi(u)$ and $\phi(v)$ are adjacent in the larger graph H. In the Colored Subgraph Isomorphism problem, the input contains a (not necessarily proper) coloring of the vertices of H and G. The task is to find a subgraph mapping ϕ that satisfies the additional constraint that for every $v \in V(G)$, the color of $\phi(v)$ has to be the same as the color v. Partitioned Subgraph Isomorphism is a special case of the colored version where every vertex of the smaller graph G has a distinct color (that is, we can assume that $V(G)$

is the set of colors). In other words, the vertices of H are partitioned into $|V(G)|$ classes, and the image of each $v \in V(G)$ is restricted to a distinct class of the partition.

It is not hard to observe that Partitioned Subgraph Isomorphism is essentially the same as binary CSP. We can reduce an instance $I = (V, D, C)$ of binary CSP to Partitioned Subgraph Isomorphism the following way. Let G be the primal graph of I. We construct a graph H, whose vertex set is $V(G) \times D$, and the color of $(v, d) \in V(G) \times D$ is v. For every constraint $\langle (u, v), R_{uv} \rangle \in C$ and every pair $(d_u, d_v) \in R_{uv}$, we add an edge connecting (u, d_u) and (v, d_v) to H.

Suppose that $f : V \to D$ is a satisfying assignment of I and consider the mapping $\phi(v) = (v, f(v))$ for every $v \in V(G)$. It is clear that ϕ respects the colors and it is a subgraph mapping: if u and v are adjacent in G, then there is a corresponding constraint $\langle (u, v), R_{uv} \rangle \in C$, and the fact that $(f(u), f(v)) \in R_{uv}$ implies that $\phi(u)$ and $\phi(v)$ are adjacent. On the other hand, suppose that ϕ is a subgraph mapping respecting the colors. This means the first coordinate of $\phi(v)$ is v; let $f(v)$ be the second coordinate of $\phi(v)$. It is straightforward to verify that f is a satisfying assignment: for every constraint $\langle (u, v), R_{uv} \rangle \in C$, vertices u and v are adjacent in G by the definition of the primal graph, and hence the fact that $(u, f(u))$ and $(v, f(v))$ are adjacent implies that $(f(u), f(v)) \in R_{uv}$.

The reduction from binary CSP to Partitioned Subgraph Isomorphism implies that any lower bound for the former problem can be transfered to the latter. Thus Theorem 3.2 implies the following result:

Corollary 3.17. *If there is a class $\mathcal{G}$ of graphs with unbounded treewidth, an algorithm $\mathbb{A}$, and an arbitrary function f such that $\mathbb{A}$ correctly decides every instance of* Partitioned Subgraph Isomorphism *and the running time is $f(G)n^{o(\mathrm{tw}(G)/\log \mathrm{tw}(G))}$ for instances with the smaller graph G in $\mathcal{G}$, then ETH fails.*

It is known that there are infinite recursively enumerable classes $\mathcal{G}$ of graphs such that for every $G \in \mathcal{G}$, both the treewidth and the number of edges are $\Theta(|V(G)|)$: for example, explicit constructions of bounded-degree expanders give such classes (cf. [125]). Using this class $\mathcal{G}$ in Corollary 3.17, we get

Corollary 3.18. *If* Partitioned Subgraph Isomorphism *can be solved in time $f(G)n^{o(k/\log k)}$, where f is an arbitrary function and k is the number of* edges *of the smaller graph G, then ETH fails.*

CHAPTER 4

Fractional edge covers, Constraint Satisfaction Problems, and database queries

As we have seen in Chapter 3, the complexity of binary CSP is very tightly characterized by the treewidth of the primal graph. This is no longer the case for CSP instances where constraints can have arbitrary arity. For example, if we consider the class $\mathcal{H}_1$ of hypergraphs containing every hypergraph H where there is an edge $e \in E(H)$ covering the set $V(H)$ of vertices, then CSP($\mathcal{H}_1$) is polynomial-time solvable (see Section 1.4). More generally, if we consider hypergraphs where the set of vertices can be covered by a constant number c of hyperedges, then the problem is still polynomial-time solvable. These simple observations crucially rely on the assumption that the constraints are represented by explicitly listing every satifying tuple.

The main contribution of this chapter is realizing that polynomial-time solvability can be achieved even if the vertices can be covered *fractionally* by c hyperedges. This follows from observing that the fractional edge cover number gives an upper bound on the maximum possible number of solutions. We complement this by showing that this bound is essentially tight. The tight bound on the number of possible solutions has particular relevance for evaluating the number of possible answers for a given database query. Despite answering a very fundamental question, it is surprising that such a bound was not known before in the literature and our result was the starting point for work by other researchers on this topic [114, 143, 157, 160, 161, 195, 196, 229]

Publications. This chapter is based on two publications. Section 4.2 of this chapter is based on the first half of an articled that appeared in *ACM Transactions on Algorithms* [126] (an extended abstract appeared in the proceedings of the SODA 2006 conference [124]). It is joint work with Martin Grohe; both authors contributed equally to the publications. Sections 4.3–4.5 of this chapter are based on the first half of an article that appeared in *SIAM Journal on Computing* [21] (an extended abstract appeared in the proceedings of the FOCS 2008 conference [20]). It is joint work with Albert Atserias and Martin Grohe, all three authors contributed equally to the publications.

4.1 Introduction

The *hypergraph* of an instance (V, D, C) has V as its vertex set and for every constraint in C a hyperedge that consists of all variables occurring in the constraint. For a class $\mathcal{H}$ of hypergraphs, we let CSP($\mathcal{H}$) be the class of all instances whose hypergraph is contained in $\mathcal{H}$. The central question is for which classes $\mathcal{H}$ of hypergraphs the problem CSP($\mathcal{H}$) is tractable. Recall from Chapter 3

(Theorem 3.1) that the corresponding question for the primal graphs (instead of hypergraphs) of instances, in which two variables are incident if they appear together in a constraint, has been completely answered in [123, 127] (under the complexity theoretic assumption FPT $\neq$ W[1]): For a class $\mathcal{G}$ of graphs, the corresponding problem CSP($\mathcal{G}$) is in polynomial time if and only if $\mathcal{G}$ has bounded treewidth. This can be generalized to CSP($\mathcal{H}$) for classes $\mathcal{H}$ of hypergraphs of *bounded hyperedge size* (that is, classes $\mathcal{H}$ for whx'ich $\max\{|e| \mid \exists\, H = (V, E) \in \mathcal{H} :\ e \in E\}$ exists). It follows easily from the results of [123, 127] that for all classes $\mathcal{H}$ of bounded hyperedge size,

$$\text{CSP}(\mathcal{H}) \in \text{PTIME} \iff \mathcal{H} \text{ has bounded treewidth} \tag{4.1}$$

(under the assumption FPT $\neq$ W[1]).

It is known that (1) does not generalize to arbitrary classes $\mathcal{H}$ of hypergraphs, see the following example.

Example 4.1. Let $\mathcal{H}_1$ be that class of all hypergraphs H that have a hyperedge that contains all vertices, that is, $V(H) \in E(H)$. Clearly, $\mathcal{H}_1$ has unbounded treewidth, because the hypergraph $(V, \{V\})$ has treewidth $|V| - 1$. We claim that CSP($\mathcal{H}_1$) $\in$ PTIME.

To see this, let $I = (V, D, C)$ be an instance of CSP($\mathcal{H}_1$). Let $\langle (v_1, \ldots, v_k), R\rangle$ be a constraint in C with $\{v_1, \ldots, v_k\} = V$. Such a constraint exists because $H_I \in \mathcal{H}_1$. Each tuple $\bar{d} = (d_1, \ldots, d_k) \in R$ completely specifies an assignment $\alpha_{\bar{d}}$ defined by $\alpha_{\bar{d}}(v_i) = d_i$ for $1 \leq i \leq k$. If for some i, j we have $v_i = v_j$, but $d_i \neq d_j$, we leave $\alpha_{\bar{d}}$ undefined.

Observe that I is satisfiable if and only if there is a tuple $\bar{d} \in R$ such that $\alpha_{\bar{d}}$ is (well-defined and) a solution for I. As $|R| \leq \|I\|$, this can be checked in polynomial time.

For some time, the largest known family of classes of hypergraphs for which CSP($\mathcal{H}$) is in PTIME consisted of all classes of bounded *hypertree width* [115–117]. Hypertree width is a hypergraph invariant that generalizes acyclicity [27, 95, 233]. It is a very robust invariant; up to a constant factor it coincides with a number of other natural invariants that measure the global connectivity of a hypergraph [8]. On classes of bounded hyperedge size, bounded hypertree width coincides with bounded tree width, but in general it does not. It has been asked in [57, 66, 113, 123] whether there are classes $\mathcal{H}$ of unbounded hypertree width such that CSP($\mathcal{H}$) $\in$ PTIME. We give an affirmative answer to this question in this section by showing that classes of hypergraphs with bounded fractional edge cover number also lead to polynomial-time solvable CSP problems. In Chapter 5, we further generalize this to classes of hypergraphs with bounded fractional hypertree width.

Our key result states that CSP($\mathcal{H}$) $\in$ PTIME for all classes $\mathcal{H}$ of bounded *fractional edge cover number*. A *fractional edge cover* of a hypergraph $H = (V, E)$ is a mapping $x : E \to [0, \infty)$ such that $\sum_{e \in E, v \in e} x(e) \geq 1$ for all $v \in V$. The number $\sum_{e \in E} x(e)$ is the *weight* of x. The *fractional edge cover number* $\rho^*(H)$ of H is the minimum of the weights of all fractional edge covers of H. It follows from standard linear programming results that this minimum exists and is rational. Furthermore, it is easy to construct classes $\mathcal{H}$ of hypergraphs that have bounded fractional edge cover number and unbounded hypertree width (see Example 5.2).

An optimal query evaluation algorithm matching this tight bound was given by Ngo et al. [195] and by Veldhuizen [229]. The bound was generalized to the context of conjunctive queries with functional dependencies by Gottlob et al. [114].

4.2 A Polynomial-time algorithm for CSPs with bounded fractional cover number

In this section we prove that if the hypergraph H_I of a CSP instance I has fractional edge cover number $\rho^*(H_I)$, then it can be decided in $\|I\|^{\rho^*(H_I)+O(1)}$ time whether I has a solution. Thus if $\mathcal{H}$

is a class of hypergraphs with bounded fractional edge cover number (that is, there is a constant r such that $\rho^*(H) \leq r$ for every $H \in \mathcal{H}$), then $\mathrm{CSP}(\mathcal{H}) \in \mathrm{PTIME}$. Actually, we prove a stronger result: A CSP instance I has at most $\|I\|^{\rho^*(H_I)}$ solutions and all the solutions can be enumerated in time $\|I\|^{\rho^*(H_I)+O(1)}$. Optimizing the extra $O(1)$ term in the exponent was not the focus of this work. But inspired by our results, optimal algorithms with running time $O(\|I\|^{\rho^*(H_I)})$ were designed later [195, 229].

Our proof relies on a combinatorial lemma known as Shearer's Lemma. We use Shearer's Lemma to bound the number of solutions of a CSP instance; our argument resembles an argument that Friedgut and Kahn [109] used to bound the number of subhypergraphs of a certain isomorphism type in a hypergraph. The author of this dissertation applied similar ideas in a completely different algorithmic context [182].

The *entropy* of a random variable X with range U is

$$h[X] := -\sum_{x \in U} \Pr(X = x) \log \Pr(X = x)$$

Shearer's lemma gives an upper bound of a distribution on a product space in terms of its marginal distributions.

Lemma 4.2 (Shearer's Lemma [63])**.** *Let $X = (X_i \mid i \in I)$ be a random variable, and let A_j, for $j \in [m]$, be (not necessarily distinct) subsets of the index set I such that each $i \in I$ appears in at least q of the sets A_j. For every $B \subseteq I$, let $X_B = (X_i \mid i \in B)$. Then*

$$\sum_{j=1}^{m} h[X_{A_j}] \geq q \cdot h[X].$$

Lemma 4.2 is easy to see in the special case when $q = 1$ and $\{A_1, \ldots, A_p\}$ is a partition of V. The proof of the general case in [63] is based on the submodularity of entropy. See also [206] for a simple proof.

Lemma 4.3. *If $I = (V, D, C)$ is a CSP instance where every constraint relation contains at most N tuples, then I has at most $N^{\rho^*(H_I)} \leq \|I\|^{\rho^*(H_I)}$ solutions.*

Proof. Let x be a fractional edge cover of H_I with $\sum_{e \in E(H_I)} x(e) = \rho^*(H_I)$; it follows from the standard results of linear programming that such an x exists with rational values. Let p_e and q be nonnegative integers such that $x(e) = p_e/q$. Let $m = \sum_{e \in E(H_I)} p_e$, and let $A_1, \ldots, A_m$ be a sequence of subsets of V that contains precisely p_e copies of the set e, for all $e \in E(H_I)$. Then every variable $v \in V$ is contained in at least

$$\sum_{e \in E(H_I): v \in e} p_e \;=\; q \cdot \sum_{e \in E(H_I): v \in e} x(e) \;\geq\; q$$

of the sets A_i (as x is a fractional edge cover). Let $X = (X_v \mid v \in V)$ be uniformly distributed on the solutions of I, which we assume to be non-empty as otherwise the claim is obvious. That is, if we denote by S the number of solutions of I, then we have $\Pr(X = \alpha) = 1/S$ for every solution α of I. Then $h[X] = \log S$. We apply Shearer's Lemma to the random variable X and the sequence A_1, $\ldots$, A_m of subsets of V. Assume that A_i corresponds to some constraint $\langle (v'_1, \ldots, v'_k), R \rangle$. Then the marginal distribution of X on $(v'_1, \ldots, v'_k)$ is 0 on all tuples not in R. Hence the entropy of X_{A_i} is is bounded by the entropy of the uniform distribution on the tuples in R, that is, $h[X_{A_i}] \leq \log N$. Thus by Shearer's Lemma, we have

$$\sum_{e \in E(H_I)} p_e \cdot \log N \geq \sum_{e \in E(H_I)} p_e \cdot h[X_e] = \sum_{i=1}^{m} h[X_{A_i}] \geq q \cdot h[X] = q \cdot \log S.$$

It follows that

$$S \leq 2^{\sum_{e \in E(H_I)} (p_e/q) \cdot \log N} = 2^{\rho^*(H_I) \cdot \log N} = N^{\rho^*(H_I)}.$$

□

We would like to turn the upper bound of Lemma 4.3 into an algorithm enumerating all the solutions, but the proof of Shearer's Lemma is not algorithmic. However, a very simple algorithm can enumerate the solutions, and Lemma 4.3 can be used to bound the running time of this algorithm. Starting with a trivial subproblem consisting only of a single variable, the algorithm enumerates all the solutions for larger and larger subproblems by adding one variable at a time. To define these subproblems, we need the following definitions:

Definition 4.4. Let R be an r-ary relation over a set D. For $1 \leq i_1 < \cdots < i_\ell \leq r$, the *projection of* R onto the components $i_1, \ldots, i_\ell$ is the relation $R^{|i_1,\ldots,i_\ell}$ which contains an ℓ-tuple $(d'_1, \ldots, d'_\ell) \in D^\ell$ if and only if there is a k-tuple $(d_1, \ldots, d_k) \in R$ such that $d'_j = d_{i_j}$ for $1 \leq j \leq \ell$.

Intuitively, a tuple is in $R^{|i_1,\ldots,i_\ell}$ if it can be extended into a tuple in R.

Definition 4.5. Let $I = (V, D, C)$ be a CSP instance and let $V' \subseteq V$ be a nonempty subset of variables. The *CSP instance* $I[V']$ *induced by* V' is $I' = (V', D, C')$, where C' is defined in the following way: for each constraint $c = \langle (v_1, \ldots, v_k), R \rangle$ having at least one variable in V', there is a corresponding constraint c' in C'. Suppose that $v_{i_1}, \ldots, v_{i_\ell}$ are the variables among $v_1, \ldots, v_k$ that are in V'. Then the constraint c' is defined as $\langle (v_{i_1}, \ldots, v_{i_\ell}), R^{|i_1,\ldots,i_\ell} \rangle$, that is, the relation is the projection of R onto the components $i_1, \ldots, i_\ell$.

Thus an assignment α on V' satisfies $I[V']$ if for each constraint c of I, there is an assignment extending α that satisfies c (however, it is not necessarily true that there is an assignment extending α that satisfies every constraint of I simultaneously). Note that that the hypergraph of the induced instance $I[V']$ is exactly the induced subhypergraph $H_I[V']$.

Theorem 4.6. *The solutions of a CSP instance I can be enumerated in time $\|I\|^{\rho^*(H_I)+O(1)}$.*

Proof. Let $V = \{v_1, \ldots, v_n\}$ be an arbitrary ordering of the variables of I and let V_i be the subset $\{v_1, \ldots, v_i\}$. For $i = 1, 2, \ldots, n$, the algorithm creates a list L_i containing the solutions of $I[V_i]$. Since $I[V_n] = I$, the list L_n is exactly what we want.

For $i = 1$, the instance $I[V_i]$ has at most $|D|$ solutions, hence the list L_i is easy to construct. Notice that a solution of $I[V_{i+1}]$ induces a solution of $I[V_i]$. Therefore, the list L_{i+1} can be constructed by considering the solutions in L_i, extending them to the variable v_{i+1} in all the $|D|$ possible ways, and checking whether this assignment is a solution of $I[V_{i+1}]$. Clearly, this can be done in $|L_i| \cdot |D| \cdot \|I[V_{i+1}]\|^{O(1)} = |L_i| \cdot \|I\|^{O(1)}$ time. By repeating this procedure for $i = 1, 2, \ldots, n-1$, the list L_n can be constructed.

The total running time of the algorithm can be bounded by $\sum_{i=1}^{n-1} |L_i| \cdot \|I\|^{O(1)}$. Observe that $\rho^*(H_{I[V_i]}) \leq \rho^*(H_I)$: $H_{I[V_i]}$ is the subhypergraph of H_I induced by V_i, thus any fractional cover of the hypergraph of I gives a fractional cover of $I[V_i]$ (for every edge $e \in E(H_{I[V_i]})$, we set the weight of e to be the sum of the weight of the edges $e' \in E(H_I)$ with $e' \cap V_i = e$). Therefore, by Lemma 4.3, $|L_i| \leq \|I\|^{\rho^*(H_I)}$, and it follows that the total running time is $\|I\|^{\rho^*(H_I)+O(1)}$. □

We note that the algorithm of Theorem 4.6 does not actually need a fractional edge cover: the fact that the hypergraph has small fractional edge cover number is used only in proving the time bound of the algorithm.

Corollary 4.7. *Let $\mathcal{H}$ be a class of hypergraphs of bounded fractional edge cover number. Then* $\mathrm{CSP}(\mathcal{H})$ *is in polynomial time.*

We conclude this section by pointing out that Lemma 4.3 is tight: there are arbitrarily large instances I where every constraint relation contains at most N tuples and the number of solutions is exactly $N^{\rho^*(H_I)}$.

Theorem 4.8. *Let H be a hypergraph. For every $N_0 \geq 1$, there is a CSP instance $I = (V, D, C)$ with hypergraph H where every constraint relation contains at most $N \geq N_0$ tuples and I has at least $N^{\rho^*(H)}$ solutions.*

Proof. A *fractional independent set* of hypergraph H is an assignment $y : V(H) \to [0, 1]$ such that $\sum_{v \in e} y(v) \leq 1$ for every $e \in E(H)$. The *weight* of y is $\sum_{v \in V(H)} y(H)$. The *fractional independent set number* $\alpha^*(H)$ is the maximum weight of a fractional independent set of H. It is a well-known consequence of linear-programming duality that $\alpha^*(H) = \rho^*(H)$ for every hypergraph H, since the two values can be expressed by a pair of primal and dual linear programs [214, Section 30.10].

Let y be a fractional independent set of weight $\alpha^*(H)$. By standard results of linear programming, we can assume that y is rational, that is, there is an integer $q \geq 1$ such that for every $v \in V(H)$, $y(v) = p_v/q$ for some nonnegative integer p_v. We define a CSP instance $I = (V, D, C)$ with $V = V(H)$ and $D = [N_0^q]$ such that for every $e \in E(H)$ where $e = \{v_1, \ldots, v_r\}$, there is a constraint $\langle (v_1, \ldots, v_r), R_e \rangle$ with

$$R_e = \{(a_1, \ldots, a_r) \mid a_i \in [N_0^{p_v}] \text{ for every } 1 \leq i \leq r\}.$$

Let $N = N_0^q$. We claim that R_e contains at most N tuples. Indeed, the number of tuples in R_e is exactly

$$\prod_{v \in e} N_0^{p_v} = N_0^{\sum_{v \in e} p_v} = N_0^{q \cdot \sum_{v \in e} p_v/q} = (N_0^q)^{\sum_{v \in e} y(v)} \leq N_0^q = N,$$

since y is a fractional independent set. Observe that $\alpha : V(H) \to D$ is a solution if and only if $\alpha(v) \in [N_0^{p_v}]$ for every $v \in V(H)$. Hence the number of solutions is exactly

$$\prod_{v \in V(H)} N_0^{p_v} = N_0^{\sum_{v \in V(H)} p_v} = N_0^{q \cdot \sum_{v \in V(H)} p_v/q} = (N_0^q)^{\alpha^*(H)} = N^{\alpha^*(H)} = N^{\rho^*(H)},$$

as required. □

The significance of this result is that it shows that there is no "better" measure than fractional edge cover number that guarantees a polynomial bound on the number of solutions, in the following formal sense. Let $w(H)$ be a width measure that guarantees a polynomial bound: that is, if I is a CSP instance where every relation has at most N tuples, then I has at most $N^{w(H)}$ solutions for some function f. Then by Theorem 4.8, we have $\rho^*(H) \leq w(H)$. This means the upper bound on the number of solutions given by $w(H)$ already follows from the bound given by Lemma 4.3 and hence $\rho^*(H)$ can be considered a stronger measure.

4.3 Relational joins

The join operation is one of the core operations of relational algebra, which in turn is the core of the standard database query language SQL. The two key components of a database system executing SQL-queries are the query optimiser and the execution engine. The optimiser translates the query

into several possible execution plans, which are basically terms of the relational algebra (also called operator trees) arranging the operations that have to be carried out in a tree-like order. Using statistical information about the data, the optimiser estimates the execution cost of the different plans and passes the best one on to the execution engine, which then executes the plan and computes the result of the query. See [54] for a survey of query optimisation techniques.

Among the relational algebra operations, joins are usually the most costly, simply because a join of two relations, just like a Cartesian product of two sets, may be much larger than the relations. Therefore, query optimisers pay particular attention to the execution of joins, especially to the execution order of sequences of joins, and to estimating the size of joins. In this chapter, we address the very fundamental questions of how to estimate the size of a sequence of joins and how to execute the sequence best from a theoretical point of view. While these questions have been intensely studied in practice, and numerous heuristics and efficiently solvable special cases are known (see, e.g., [54, 110, 119]), the very basic theoretical results we present here and their consequences apparently have not been noticed so far. Our key starting observation is that the size of a sequence of joins is tightly linked to two combinatorial parameters of the underlying database schema, the *fractional edge cover number*, and the *maximum density*.

To make this precise, we need to get a bit more technical: A *join query* Q is an expression of the form

$$R_1(a_{11}, \ldots, a_{1r_1}) \bowtie \cdots \bowtie R_m(a_{m1}, \ldots, a_{mr_m}), \tag{4.2}$$

where the R_i are *relation names* with *attributes* $a_{i1}, \ldots, a_{ir_i}$. Let A be the set of all attributes occurring in Q and $n = |A|$. A *database instance* D for Q consists of relations $R_i(D)$ of arity r_i. It is common to think of the relation $R_i(D)$ as a table whose columns are labelled by the attributes $a_{i1}, \ldots, a_{ir_i}$ and whose rows are the tuples in the relation. The *answer*, or *set of solutions*, of the query Q in D is the n-ary relation $Q(D)$ with attributes A consisting of all tuples t whose projection on the attributes of R_i belongs to the relation $R_i(D)$, for all i. Hence we are considering *natural joins* here (all of our results can easily be transferred to *equi-joins*, but not to general *θ-joins*). Now the most basic question is how large $Q(D)$ can get in terms of the size of the database $|D|$, or more generally, in terms of the sizes of the relations R_i. We address this question both in the worst case and the average case, and also subject to various constraints imposed on D.

Example 4.9. At this point a simple example would probably help to understand what we are after. Let $R(a, b)$, $S(b, c)$ and $T(c, a)$ be three relations on the attributes a, b and c. Consider the join query

$$Q(a, b, c) := R(a, b) \bowtie S(b, c) \bowtie T(c, a).$$

The answer of Q is precisely the set of triples (u, v, w) such that $(u, v) \in R$, $(v, w) \in S$ and $(w, u) \in T$. How large can the answer size of Q get as a function of $|R|$, $|S|$ and $|T|$? First note that a trivial upper bound is $|R| \cdot |S| \cdot |T|$. However one quickly notices that an improved bound can be derived from the fact that the relations in Q have overlapping sets of attributes. Indeed, since any solution for any pair of relations in Q determines the solution for the third, the answer size of Q is bounded by $\min\{|R| \cdot |S|, |S| \cdot |T|, |T| \cdot |R|\}$. Now, is this the best general upper bound we can get as a function of $|R|$, $|S|$ and $|T|$? As it turns out, it is not. Although not obvious, it will follow from the results in this chapter that the optimal upper bound in this case is $\sqrt{|R| \cdot |S| \cdot |T|}$: the answer size of Q is always bounded by this quantity, and for certain choices of the relations R, S, T, this upper bound is achieved.

Besides estimating the answer size of join queries, we also study how to exploit this information to actually compute the query. An *execution plan* for a join query describes how to carry out the evaluation of the query by simple operations of the relational algebra such as joins of two relations

or projections. The obvious execution plans for a join query break up the sequence of joins into pairwise joins and arrange these in a tree-like fashion. We call such execution plans *join plans*. As described in [54], most practical query engines simply arrange the joins in some linear (and not even a tree-like) order and then evaluate them in this order. However, it is also possible to use other operations, in particular projections, in an execution plan for a join query. We call execution plans that use joins and projections *join-project plans*. It is one of our main results that, even though projections are not necessary to evaluate join queries, their use may speed up the evaluation of a query super-polynomially.

Fractional covers, worst-case size, and join-project plans. Recall that an *edge cover* of a hypergraph H is a set C of edges of H such that each vertex is contained in at least one edge in C, and the *edge cover number* $\rho(H)$ of H is the minimum size among all edge covers of H. A *fractional edge cover* of H is a feasible solution for the linear programming relaxation of the natural integer linear program describing edge covers, and the *fractional edge cover number* $\rho^*(H)$ of H is the cost of an optimal solution. With a join query Q of the form (4.2) we can associate a hypergraph $H(Q)$ whose vertex set is the set of all attributes of Q and whose edges are the attribute sets of the relations R_i. The *(fractional) edge cover number* of Q is defined by $\rho(Q) = \rho(H(Q))$ and $\rho^*(Q) = \rho^*(H(Q))$. Note that in Example 4.9, the hypergraph $H(Q)$ is a triangle. Therefore in that case $\rho(Q) = 2$ while it can be seen that $\rho^*(Q) = 3/2$.

An often observed fact about edge covers is that, for every given database D, the size of $Q(D)$ is bounded by $|D|^{\rho(Q)}$, where $|D|$ is the total number of tuples in D. Much less obvious is the fact that the size of $Q(D)$ can actually be bounded by $|D|^{\rho^*(Q)}$, as proved in Section 4.2 in the context (and the language) of constraint satisfaction problems. This is a consequence to Shearer's Lemma [63], which is a combinatorial consequence of the submodularity of the entropy function, and is closely related to a result due to Friedgut and Kahn [109] on the number of copies of a hypergraph in another. We observe that the fractional edge cover number $\rho^*(Q)$ also provides a lower bound to the worst-case answer size: we show that for every Q, there exist arbitrarily large databases D for which the size of $Q(D)$ is at least $(|D|/|Q|)^{\rho^*(Q)}$. The proof is a simple application of linear programming duality. Theorem 4.6 from Section 4.2 implies that for every join query there is a join-project plan, which can easily be obtained from the query and certainly be computed in polynomial time, that computes $Q(D)$ in time $|D|^{\rho^*(Q)+O(1)}$. Our lower bound shows that this is optimal up to a polynomial factor. In particular, we get the following equivalences giving an exact combinatorial characterisation of all classes of join queries that have polynomial size answers and can be evaluated in polynomial time.

Theorem 4.10. *Let $\mathcal{Q}$ be a class of join queries. Then the following statements are equivalent:*

1. *Queries in $\mathcal{Q}$ have answers of polynomial size.*
2. *Queries in $\mathcal{Q}$ can be evaluated in polynomial time.*
3. *Queries in $\mathcal{Q}$ can be evaluated in polynomial time by an explicit join-project plan.*
4. *$\mathcal{Q}$ has bounded fractional edge cover number.*

Note that a priori it is not even obvious that the first two statements are equivalent, that is, that for every class of queries with polynomial size answers there is a polynomial time evaluation algorithm (the converse, of course, is trivial).

Hence with regard to worst-case complexity, join-project plans are optimal (up to a polynomial factor) for the evaluation of join queries. Our next result is that join plans are not: We prove that there are arbitrarily large join queries Q and database instances D such that our generic join-project plan computes $Q(D)$ in at most cubic time, whereas any join plan requires time $|D|^{\Omega(\log|Q|)}$ to

compute $Q(D)$. We also observe that this bound is tight, i.e., the ratio of the exponents between the best join plan and the best join-project plan is at most logarithmic in $|Q|$. Hence incorporating projections into a query plan may lead to a superpolynomial speed-up even if the projections are completely irrelevant for the query answer.

Size and integrity constraints. So far, we considered worst-case bounds which make no assumptions on the database. However, practical query optimisers usually exploit additional information about the databases when computing their size estimates. We consider the simplest such setting where the sizes of the relations are known (called histograms in the database literature), and we want to get a (worst case) estimate on the size of $Q(D)$ subject to the constraint that the relations in D have the given sizes.

By suitably modifying the objective function of the linear program for edge covers, we obtain results analogous to those obtained for the unconstrained setting. A notable difference between the two results is that here the gap between upper and lower bound becomes 2^{-n}, where n is the number of attributes, instead of $|Q|^{-\rho^*}$. We give an example showing that the gap between upper and lower bound is essentially tight. However, this is not an inadequacy of our approach through fractional edge covers, but due to the inherent complexity of the problem: by a reduction from the maximum independent-set problem on graphs, we show that, unless NP = ZPP, there is no polynomial time algorithm that approximates the worst case answer size $|Q(D)|$ for given Q and relation sizes N_R by a better-than-exponential factor.

Preliminaries on database queries. For integers $m \leq n$, by $[m, n]$ we denote the set $\{m, m+1, \ldots, n\}$ and by $[n]$ we denote $[1, n]$. All our logarithms are base 2.

Our terminology is similar to that used in [4]: An *attribute* is a symbol a with an associated *domain* $\mathrm{dom}(a)$. If not specified otherwise, we assume $\mathrm{dom}(a)$ to be an arbitrary countably infinite set, say, $\mathbb{N}$. Sometimes, we will impose restrictions on the size of the domains. A *relation name* is a symbol R with an associated finite set of attributes A. For a set $A = \{a_1, \ldots, a_n\}$ of attributes, we write $R(A)$ or $R(a_1, \ldots, a_n)$ to denote that A is the set of attributes of R. The *arity* of $R(A)$ is $|A|$. A *schema* is a finite set of relation names. If $\sigma = \{R(A_1), \ldots, R(A_m)\}$, we write A_σ for $\bigcup_i A_i$.

For a set A of attributes, an *A-tuple* is a mapping t that associates an element $t(a)$ from $\mathrm{dom}(a)$ with each $a \in A$. Occasionally, we denote A-tuples in the form $t = (t_a : a \in A)$, with the obvious meaning that t is the A-tuple with $t(a) = t_a$. The set of all A-tuples is denoted by $\mathrm{tup}(A)$. An *A-relation* is a set of A-tuples. The *active domain* of an A-relation R is the set $\{t(a) : t \in R, a \in A\}$. The *projection* of an A-tuple t to a subset $B \subseteq A$ is the restriction $\pi_B(t)$ of t to B, and the *projection* of an A-relation R is the set $\pi_B(R) = \{\pi_B(t) : t \in R\}$.

A *database instance* D of schema σ, or a *σ-instance*, consists of an A-relation $R(D)$ for every relation name R in σ with set of attributes A. The *active domain* of D is the union of active domains of all its relations. The *size* of a σ-instance D is $|D| := \sum_{R \in \sigma} |R(D)|$.

A *join query* is an expression

$$Q := R_1(A_1) \bowtie \cdots \bowtie R_m(A_m),$$

where R_i is a relation name with attributes A_i. The *schema* of Q is the set $\{R_1, \ldots, R_m\}$, and the *set of attributes* of Q is $\bigcup_i A_i$. We often denote the set of attributes of a join query Q by A_Q, and we write $\mathrm{tup}(Q)$ instead of $\mathrm{tup}(A_Q)$. The *size* of Q is $|Q| := \sum_i |A_i|$. We write $H(Q)$ for the (multi-)hypergraph that has vertex-set A_Q and edge-(multi-)set $\{A_1, \ldots, A_m\}$. If D is an $\{R_1, \ldots, R_m\}$-instance, the *answer* of Q on D is the A_Q-relation

$$Q(D) = \big\{t \in \mathrm{tup}(A_Q) : \pi_{A_i}(t) \in R_i(D) \text{ for every } i \in [m]\big\}.$$

A *join plan* is a term built from relation names and binary join operators. For example, $(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$ and $((R_1 \bowtie R_2) \bowtie R_3) \bowtie (R_1 \bowtie R_4)$ are two join plans corresponding to the

same join query $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. A *join-project plan* is a term built from relation names, binary join operators, and unary project operators. For example, $(\pi_A(R_1) \bowtie R_2) \bowtie \pi_B(R_1)$ is a join-project plan. Join-project plans have a natural representation as labelled binary trees, where the leaves are labelled by relation names, the unary nodes are labelled by projections π_A, and the binary nodes by joins. Evaluating a join plan or join-project plan φ in a database instance D means substituting the relation names by the actual relations from D and carrying out the operations in the expression. We denote the resulting relation by $\varphi(D)$. A join(-project) plan φ is a plan *for* a query Q if $\varphi(D) = Q(D)$ for every database D. The *subplans* of a join(-project) plan are defined in the obvious way. For example, the subplans of $(R_1 \bowtie R_2) \bowtie \pi_A(R_3 \bowtie R_4)$ are R_1, R_2, R_3, R_4, $R_1 \bowtie R_2$, $R_3 \bowtie R_4$, $\pi_A(R_3 \bowtie R_4)$, $(R_1 \bowtie R_2) \bowtie \pi_A(R_3 \bowtie R_4)$. If φ is a join project plan, then we often use A_φ to denote the set of attributes of the query computed by φ (this only includes "free" attributes and not those projected away by some projection in φ), and we write $\mathrm{tup}(\varphi)$ instead of $\mathrm{tup}(A_\varphi)$.

4.4 Size bounds on database queries

Let Q be a join query with schema σ. For every $R \in \sigma$, let A_R be the set of attributes of R, so that $A_\sigma = \bigcup_R A_R$. The fractional edge covers are precisely the feasible solutions $(x_R : R \in \sigma)$ for the following linear program L_Q, and the fractional edge cover number $\rho^*(Q)$ is the cost of an optimal solution.

$$\begin{array}{lll} L_Q: \text{ minimise} & \sum_R x_R & \\ \text{subject to} & \sum_{R\,:\,a \in A_R} x_R \geq 1 & \text{for all } a \in A_\sigma, \\ & x_R \geq 0 & \text{for all } R \in \sigma. \end{array} \tag{4.3}$$

By standard arguments, there always is an optimal fractional edge cover whose values are rational and of bit-length polynomial in $|Q|$. By restating Lemma 4.3 in the context of database queries, we can observe that fractional edge covers can be used to give an upper bound on the size of a query.

Lemma 4.11. *Let Q be a join query with schema σ and let D be a σ-instance. Then for every fractional edge cover $(x_R : R \in \sigma)$ of Q we have*

$$|Q(D)| \leq \prod_{R \in \sigma} |R(D)|^{x_R} = 2^{\sum_{R \in \sigma} x_R \log |R_D|}.$$

Note that the fractional edge cover in the statement of the lemma is not necessarily one of minimum cost. For the reader's convenience, we give a proof of this lemma here as well, using the language of database queries.

Proof of Lemma 4.11. Let A_R be the set of attributes of $R \in \sigma$ so that $A_\sigma = \bigcup_R A_R$. Without loss of generality we may assume that the fractional edge cover x_R only takes rational values, because the rationals are dense in the reals. Let p_R and q be nonnegative integers such that $x_R = p_R/q$. Let $m = \sum_R p_R$, and let $A_1, \ldots, A_m$ be a sequence of subsets of A_σ that contains precisely p_R copies of the set A_R, for all $R \in \sigma$. Then every attribute $a \in A_\sigma$ is contained in at least q of the sets A_i, because

$$\big|\{i \in [m] : a \in A_i\}\big| = \sum_{R:a \in A_R} p_R \;=\; q \cdot \sum_{R:a \in A_R} x_R \;\geq\; q.$$

Let $X = (X_a \mid a \in A_\sigma)$ be uniformly distributed on $Q(D)$, which we assume to be non-empty as otherwise the claim is obvious. That is, for every tuple $t \in Q(D)$ we have $\Pr[X = t] = 1/|Q(D)|$, and for all other A-tuples we have $\Pr[X = t] = 0$. Then $h[X] = \log |Q(D)|$. We apply Shearer's

Lemma to the random variable X and the sets A_R, for $R \in \sigma$. (Thus we have $I = A_\sigma$ and $J = \sigma$.) Note that for every $R \in \sigma$ the marginal distribution of X on A_R is 0 on all tuples not in $R(D)$. Hence the entropy of X_{A_R} is bounded by the entropy of the uniform distribution on $R(D)$, that is, $h[X_{A_R}] \leq \log |R(D)|$. Thus by Shearer's Lemma, we have

$$\sum_{R \in \sigma} p_R \cdot \log |R(D)| \geq \sum_{R \in \sigma} p_R h[X_{A_R}] = \sum_{i=1}^{m} h[X_{A_i}] \geq q \cdot h[X] = q \cdot \log |Q(D)|.$$

It follows that

$$|Q(D)| \leq 2^{\sum_{R \in \sigma} (p_R/q) \cdot \log |R(D)|} = \prod_{R \in \sigma} |R(D)|^{x_R}.$$

□

The next lemma shows that the upper bound of the previous lemma is tight. Again, it is a restatement of Theorem 4.8 in the language of database queries.

Lemma 4.12. *Let Q be a join query with schema σ, and let $(x_R : R \in \sigma)$ be an optimal fractional edge cover of Q. Then for every $N_0 \in \mathbb{N}$ there is a σ-instance D such that $|D| \geq N_0$ and*

$$|Q(D)| \geq \prod_{R \in \sigma} |R(D)|^{x_R}.$$

Furthermore, we can choose D in such a way that $|R(D)| = |R'(D)|$ for all $R, R' \in \sigma$ with $x_R, x_{R'} > 0$.

Proof. Let A_R be the set of attributes of $R \in \sigma$ so that $A_\sigma = \bigcup_R A_R$. Recall $(x_R : R \in \sigma)$ is an optimal solution for the linear program (4.3). By LP-duality, there is a solution $(y_a : a \in A_\sigma)$ for the dual linear program

$$\begin{array}{lll} \text{maximise} & \sum_a y_a & \\ \text{subject to} & \sum_{a \in A_R} y_a \leq 1 & \text{for all } R \in \sigma, \\ & y_a \geq 0 & \text{for all } a \in A_\sigma \end{array} \tag{4.4}$$

such that $\sum_a y_a = \sum_R x_R$. There even exists such a solution with rational values.

We take an optimal solution $(y_a : a \in A_\sigma)$ with $y_a = p_a/q$, where $q \geq 1$ and $p_a \geq 0$ are integers. Let $N_0 \in \mathbb{N}$, and let $N = N_0^q$. We define a σ-instance D by letting

$$R(D) := \left\{ t \in \mathrm{tup}(A_R) : t(a) \in [N^{p_a/q}] \text{ for all } a \in A_R \right\}$$

for all $R \in \sigma$. Here we assume that $\mathrm{dom}(a) = \mathbb{N}$ for all attributes a. As there is at least one a with $y_a > 0$ and hence $p_a \geq 1$, we have $|D| \geq N^{1/q} = N_0$. Observe that

$$|R(D)| = \prod_{a \in A_R} N^{p_a/q} = N^{\sum_{a \in A_R} y_a} \leq N$$

for all $R \in \sigma$. Furthermore, $Q(D)$ is the set of all tuples $t \in \mathrm{tup}(A_\sigma)$ with $t(a) \in [N^{p_a/q}]$ for every $a \in A_\sigma$. Hence

$$|Q(D)| = \prod_{a \in A} N^{p_a/q} = N^{\sum_{a \in A_\sigma} y_a} = N^{\sum_{R \in \sigma} x_R} = \prod_{R \in \sigma} N^{x_R} \geq \prod_{R \in \sigma} |R(D)|^{x_R},$$

as required. To see that $|R(D)|$ is the same for every relation R with $x_R > 0$, we argue as follows. By complementary slackness of linear programming we have

$$\sum_{a \in A_R} y_a = 1 \quad \text{for all } R \in \sigma \text{ with } x_R > 0.$$

Thus $|R(D)| = N$ for all $R \in \sigma$ with $x_R > 0$ and

$$|Q(D)| = \prod_{R \in \sigma} N^{x_R} = \prod_{R \in \sigma} |R(D)|^{x_R}.$$

□

Now we show how Lemmas 4.11 and 4.12 give the equivalence between statements (1) and (4) of Theorem 4.10. Assume (1) and let $c > 0$ be a constant such that $|Q(D)| \leq |D|^c$ for every $Q \in \mathcal{Q}$ and every instance D. For a fixed join query $Q \in \mathcal{Q}$, if $(x_R^* : R \in \sigma)$ denotes the optimal fractional edge cover of Q, Lemma 4.12 states that there exist arbitrarily large instances D such that $|R(D)| = |D|/|\sigma|$ for every $R \in \sigma$ and

$$|Q(D)| \geq \prod_{R \in \sigma} |R(D)|^{x_R^*} \geq (|D|/|Q|)^{\sum_{R \in \sigma} x_R^*} = (|D|/|Q|)^{\rho^*(Q)}.$$

In paricular, there exist arbitrarily large instances D for which $(|D|/|Q|)^{\rho^*(Q)} \leq |D|^c$. It follows that $\rho^*(Q) \leq c$ and hence (4) in Theorem 4.10. The converse is even more direct. Assume (4) and let $c > 0$ be a constant such that $\rho^*(Q) \leq c$ for every $Q \in \mathcal{Q}$. For a fixed join query $Q \in \mathcal{Q}$, if $(x_R^* : R \in \sigma)$ denotes the optimal fractional edge cover of Q, Lemma 4.11 states that for every instance D we have

$$|Q(D)| \leq \prod_{R \in \sigma} |R(D)|^{x_R^*} \leq |D|^{\sum_{R \in \sigma} x_R^*} = |D|^{\rho^*(Q)}.$$

It follows that $|Q(D)| \leq |D|^c$ for every D and hence (1) in Theorem 4.10.

4.4.1 Execution plans

Theorem 4.6 in Section 4.2 shows that there is an algorithm for evaluating a join query Q in a database D that runs in time $|D|^{\rho^*(Q)+O(1)}$. An analysis of the proof shows that the algorithms can actually be cast as the evaluation of an explicit (and simple) join-project plan. For the reader's convenience, we give a proof of this fact here. Combined with the bounds obtained in the previous section, this yields Theorem 4.10.

We define the *size* of a k-ary relation R to be the number $||R|| := |R| \cdot k$. The bounds stated in the following fact depend on the machine model; the statement we give is based on standard random access machines with a uniform cost measure. Other models may require additional logarithmic factors.

Fact 4.13. *The following hold:*

1. *The join $R \bowtie S$ of two relations R and S can be computed in time $O(||R|| + ||S|| + ||R \bowtie S||)$.*
2. *The projection $\pi_B(R)$ of an A-relation R to a subset $B \subseteq A$ can be computed in time $O(||R||)$.*

For details and a proof of the fact, we refer the reader to [101]. The following theorem gives the promised join-project plan:

Theorem 4.14. *For every join query Q, there is a join-project plan for Q that can be evaluated in time $O\big(|Q|^2 \cdot |D|^{\rho^*(Q)+1}\big)$ on every given instance D. Moreover, there is a polynomial-time algorithm that, given Q, computes the join-project plan.*

Proof. Let $Q = R_1(A_1) \bowtie \cdots \bowtie R_m(A_m)$ be a join query and D an instance for Q. Suppose that the attributes of Q are $\{a_1, \ldots, a_n\}$. For $i \in [n]$, let $B_i := \{a_1, \ldots, a_i\}$. Furthermore, let

$$\varphi_1 := \big(\cdots(\pi_{B_1}(R_1) \bowtie \pi_{B_1}(R_2)) \bowtie \cdots \bowtie \pi_{B_1}(R_m)\big),$$
$$\varphi_{i+1} := \Big(\cdots\big((\varphi_i \bowtie \pi_{B_{i+1}}(R_1)) \bowtie \pi_{B_{i+1}}(R_2)\big) \bowtie \cdots \bowtie \pi_{B_{i+1}}(R_m)\Big) \qquad \text{for all } i \geq 1.$$

It is easy to see that for every $i \in [n]$ it holds that $\varphi_i(D) = \pi_{B_i}(Q(D))$ and hence $\varphi_n(D) = Q(D)$. Hence to compute $Q(D)$, we can evaluate the join-project plan φ_n.

To estimate the cost of the evaluating the plan, we need to establish the following claim:

For every $i \in [n]$ we have $|\varphi_i(D)| \leq |D|^{\rho^*(Q)}$.

To see this, we consider the join query

$$Q_i := {R_1}^i \bowtie \cdots \bowtie {R_m}^i,$$

where ${R_j}^i$ is a relation name with attributes $B_i \cap A_j$. The crucial observation is that $\rho^*(Q_i) \leq \rho^*(Q)$, because if $(x_R : R \in \sigma)$ is fractional edge cover of Q, then letting $x_{R^i} = x_R$ for every $R \in \sigma$ we get a fractional edge cover of Q_i of the same cost. If we let D_i be the database instance with ${R_j}^i(D_i) := \pi_{B_i}(R_j)$ for all $j \in [m]$, then we get

$$\varphi_i(D) = Q_i(D_i) \leq |D_i|^{\rho^*(Q_i)} \leq |D|^{\rho^*(Q)}.$$

This proves the claim.

We further observe that all intermediate results in the computation of $\varphi_{i+1}(D)$ from $\varphi_i(D)$ are contained in

$$\varphi_i(D) \times U,$$

where U is the active domain of D. Hence their size is bounded by $|\varphi_i(D)| \cdot |D| \leq |D|^{\rho^*(Q)+1}$, and by Fact 4.13 they can be computed in time $O(|D|^{\rho^*(Q)+1})$. Overall, we have to compute $n \cdot m$ projections, each requiring time $O(D)$, and $n \cdot m$ joins, each requiring time $O(|D|^{\rho^*(Q)+1})$. This yields the desired running time. □

We shall prove next that join plans perform significantly worse than join-project plans. Note that to evaluate a join plan one has to evaluate all its subplans. Hence for every subplan ψ of φ and every instance D, the size $|\psi(D)|$ is a lower bound for the time required to evaluate φ in D.

Theorem 4.15. *For every $m, N \in \mathbb{N}$ there are a join query Q and an instance D with $|Q| \geq m$ and $|D| \geq N$, and:*

1. *$\rho^*(Q) \leq 2$ and hence $|Q(D)| \leq |D|^2$ (actually, $|Q(D)| \leq |D|$).*
2. *Every join plan φ for Q has a subplan ψ such that $|\psi(D)| \geq |D|^{\frac{1}{5}\log|Q|}$.*

Proof. Let $n = \binom{2m}{m}$. For every $s \subseteq [2m]$ with $|s| = m$, let a_s be an attribute with domain $\mathbb{N}$. For every $i \in [2m]$, let R_i be a relation name having as attributes all a_s such that $i \in s$. Let A_i be the set of attributes of R_i and $A = \bigcup_{i \in [2m]} A_i$. The arity of R_i is

$$|A_i| = \binom{2m-1}{m-1} = \frac{m}{2m} \cdot \binom{2m}{m} = \frac{n}{2}.$$

Let $Q := R_1 \bowtie \cdots \bowtie R_{2m}$. Then $|Q| = 2m \cdot n/2 = m \cdot n$. Furthermore, $\rho^*(Q) \leq 2$. To see this, let $x_{R_i} = 1/m$ for every $i \in [2m]$. This forms a fractional edge cover of Q, because for every $s \subseteq [2m]$ with $|s| = m$, the attribute a_s appears in the m atoms R_i with $i \in s$.

Next, we define an instance D by letting $R_i(D)$ be the set of all A_i-tuples that have an arbitrary value from $[N]$ in one coordinate and 1 in all other coordinates. Formally,

$$R_i(D) := \bigcup_{a \in A_i} \bigcap_{b \in A_i \setminus a} \{t \in \mathrm{tup}(A_i) : t(a) \in [N], t(b) = 1\}.$$

Observe that $|R_i(D)| = (N-1)n/2 + 1$ for all $i \in [2m]$ and thus

$$|D| = (N-1)mn + 2m \geq N.$$

Furthermore, $Q(D)$ is the set of all A-tuples that have an arbitrary value from $[N]$ in one attribute and 1 in all other coordinates (it is not possible that two attributes have value different from 1, as every two attributes appear together in some relation). Hence $|Q(D)| = (N-1)n + 1 \leq |D|$. This completes the proof of (1).

To prove (2), we shall use the following simple (and well-known) combinatorial lemma:

Lemma 4.16. *Let T be a binary tree whose leaves are coloured with $2m$ colours, for some $m \geq 1$. Then there exists a node t of T such that at least $(m+2)/2$ and at most $m+1$ of the colours appear at leaves that are descendants of t.*

Proof. For every node t of T, let $c(t)$ be the number of colours that appear at descendants of T. The *height* of a node t is the length of the longest path from t to a leaf.

Let t be a node of minimum height such that $c(t) \geq m+2$, and let u_1, u_2 be the children of t. (Note that t cannot be a leaf because $c(t) \geq 2$.) Then $c(u_i) \leq m+1$ for $i = 1, 2$. Furthermore, $c(u_1) + c(u_2) \geq c(t)$, hence $c(u_i) \geq (m+2)/2$ for at least one i. ⌟

Continuing the proof of the theorem, we let φ be a join plan for Q. We view the term φ as a binary tree T whose leaves are labelled by atoms R_i. We view the atoms as colours. Applying the lemma, we find a node t of T such that at least $(m+2)/2$ and at most $m+1$ of the colours appear at leaves that are descendants of t. Every inner node of the tree corresponds to a subplan of φ. We let ψ be the subplan corresponding to t. Then at least $(m+2)/2$ and at most $m+1$ atoms R_i appear in ψ. By symmetry, we may assume without loss of generality that the atoms of ψ are $R_1, \ldots, R_\ell$ for some $\ell \in \left[\lceil (m+2)/2 \rceil, m+1\right]$. Hence ψ is a plan for the join query

$$R_1 \bowtie \cdots \bowtie R_\ell.$$

Let $B := \bigcup_{i=1}^{\ell} A_i$ be the set of all attributes occurring in ψ. For $i \in [m+1]$, let $s_i = \{i\} \cup [m+2, 2m]$. Then for all $i, j \in [\ell]$ we have $a_{s_i} \in A_j$ if and only if $i = j$. Hence all tuples $t \in \mathrm{tup}(B)$ with $t(a_{s_i}) \in [N]$ for all $i \in [\ell]$ and $t(b) = 1$ for all $b \in B \setminus \{a_{s_1}, \ldots, a_{s_\ell}\}$ are contained in $\psi(D)$. As there are N^ℓ such tuples, it follows that

$$|\psi(Q)| \geq N^\ell \geq N^{(m+2)/2}.$$

Statement (2) of the lemma follows, because

$$\log |Q| = \log m + \log n \leq \log m + \log 2^{2m} = \log m + 2m \leq 5 \cdot (m+2)/2,$$

provided m is large enough, which we may assume without loss of generality. □

Statement (2) of the theorem implies that any evaluation algorithm for the query Q based on evaluating join plans, which may even depend on the database instance, has a running time at least $|D|^{\Omega(\log|Q|)}$. This is to be put in contrast with the running time $O(|Q|^2 \cdot |D|^3)$ from Theorem 4.14. It is a natural question to ask if the difference can be even worse, i.e., more than logarithmic in the exponent.

Using the well-known fact that the integrality gap of the linear program for edge covers is logarithmic in the number of vertices of the hypergraph (that is, attributes of the join query), we prove below that for every query Q there is a join plan φ that can be evaluated in time $O(|Q| \cdot |D|^{2\rho^*(Q)\cdot\log|Q|})$, hence the lower bound is tight up to a small constant factor.

Proposition 4.17. *For every join query Q, there is a join plan for Q that can be evaluated in time $O(|Q| \cdot |D|^{2\rho^*(Q)\cdot\log|Q|})$ on every given instance D.*

Proof. Let Q be a join query with schema σ. For every $R \in \sigma$ let A_R be the set of attributes of R so that $A_\sigma = \bigcup_{R\in\sigma} A_R$. An *edge cover* of Q is a subset $\gamma \subseteq \sigma$ such that $A_\sigma \subseteq \bigcup_{R\in\gamma} A_R$. The *edge cover number* $\rho(Q)$ of Q is the minimum size of an edge cover for Q. Observe that edge covers correspond to $\{0,1\}$-valued fractional edge covers and that the edge cover number is precisely the cost of the optimal integral fractional edge cover. It is well known that the integrality gap for the linear program defining fractional edge covers is H_n, where $n = |A_\sigma|$ and H_n is the nth harmonic number (see, for example, [228], Chapter 13). It is known that $H_n \leq 2\log n$. Now the join plan consists in first joining the relations that form an edge cover of size $2\rho^*(Q) \cdot \log|Q|$ in arbitrary order, and then joining the result with the rest of relations in arbitrary order. □

Furthermore, the proof of Proposition 4.17 shows that, for every join query Q, there is a join plan that can be evaluated in time $O(|Q| \cdot |D|^{\rho(Q)})$, where $\rho(Q)$ denotes the edge cover number of Q. However, note that not only $|D|^{\rho(Q)}$ is potentially superpolynomial over $|D|^{\rho^*(Q)}$, but also finding this plan is in general NP-hard. Compare this with the fact that the join-project plan given by Theorem 4.14 can be found efficiently.

4.5 Size constraints

To estimate the size of joins, practical query optimisers use statistical information about the database instance such as the sizes of the relations, the sizes of some of their projections, or histograms. In this section we consider the simplest such setting where the size of the relations is known, and we prove a (worst-case) estimate on the size of $Q(D)$ subject to the constraint that the relations in D have the given sizes.

4.5.1 Size bounds under size constraints

Let Q be a join query with schema σ. For every $R \in \sigma$, let A_R be the set of attributes of R so that $A_\sigma = \bigcup_R A_R$. For every $R \in \sigma$, let N_R be a natural number, and let $L_Q(N_R : R \in \sigma)$ be the following linear program:

$$\begin{array}{lll} \text{minimise} & \sum_R x_R \cdot \log N_R & \\ \text{subject to} & \sum_{R:a\in A_R} x_R \geq 1 & \text{for all } a \in A_\sigma, \\ & x_R \geq 0 & \text{for all } R \in \sigma. \end{array} \tag{4.5}$$

Note that the only difference with L_Q as defined in (4.3) is the objective function. This implies that every feasible solution of $L_Q(N_R : R \in \sigma)$ is also a fractional edge cover of Q.

Theorem 4.18. *Let Q be a join query with schema σ and let $N_R \in \mathbb{N}$ for all $R \in \sigma$. Let n be the number of attributes of Q, and let $(x_R : R \in \sigma)$ be an optimal solution of the linear program $L_Q(N_R : R \in \sigma)$.*

1. *For every σ-instance D with $|R(D)| = N_R$ for all R it holds that $|Q(D)| \leq \prod_R N_R^{x_R}$*
2. *There is a σ-instance D such that $|R(D)| = N_R$ for all $R \in \sigma$ and $|Q(D)| \geq 2^{-n} \prod_R N_R^{x_R}$.*

Proof. Statement (1) is an immediate consequence of Lemma 4.11. To prove (2), we exploit LP duality again. The LP-dual of $L_Q(N_R : R \in \sigma)$ is the following linear program $D_Q(N_R : R \in \sigma)$:

$$\begin{array}{lll} \text{maximise} & \sum_a y_a & \\ \text{subject to} & \sum_{a \in A_R} y_a \leq \log N_R & \text{for all } R \in \sigma, \\ & y_a \geq 0 & \text{for all } a \in A_\sigma. \end{array}$$

Let $(y_a : a \in A_\sigma)$ be an optimal solution for the dual. Then $\sum_{a \in A_\sigma} y_a = \sum_{R \in \sigma} x_R \cdot \log N_R$.

For all $a \in A_\sigma$, let $y'_a = \log \lfloor 2^{y_a} \rfloor \leq y_a$. We set

$$R' := \Big\{ t \in \mathrm{tup}(A_R) : t(a) \in \big[2^{y'_a}\big] \text{ for all } a \in A_R \Big\}.$$

Then

$$|R'| = \prod_{a \in A_R} 2^{y'_a} = \prod_{a \in A_R} \lfloor 2^{y_a} \rfloor \leq 2^{\sum_{a \in A_R} y_a} \leq 2^{\log N_R} = N_R.$$

We arbitrarily add tuples to R' to obtain a relation $R(D)$ of size exactly N_R. In the resulting instance D, we have

$$|Q(D)| \geq \prod_{a \in A_\sigma} 2^{y'_a} \geq \prod_{a \in A_\sigma} \frac{2^{y_a}}{2} = 2^{-n} \cdot 2^{\sum_{a \in A_\sigma} y_a} = 2^{-n} \cdot 2^{\sum_{R \in \sigma} x_R \cdot \log N_R} = 2^{-n} \cdot \prod_{R \in \sigma} N_R^{x_R}.$$

□

Even though usually the query is much smaller than the database instance and hence we may argue that a constant factor that only depends on the size of the query is negligible, the exponential factor in the lower bound of Theorem 4.18(2) is unpleasant. In the following, we shall prove that the lower bound cannot be improved substantially. In the next example we show that we cannot replace the lower bound of Theorem 4.18(2) by $2^{-(1-\epsilon)n} \prod_R N_R^{x_R}$ for any $\epsilon > 0$. This seems to indicate that maybe the approach to estimating the size of joins through fractional edge covers is no longer appropriate in the setting where the size of the relations is fixed. However, we shall then see that, in some sense, there is no better approach. In Theorem 4.20, we shall prove that there is no polynomial time algorithm that, given a query Q and relation sizes N_R, for $R \in \sigma$, approximates the worst case size of the query answer to a factor better than $2^{n^{1-\epsilon}}$.

Example 4.19. We give an example where $\prod_{R \in \sigma} N_R^{x_R}$ is roughly 2^n but $|Q(D)|$ is at most $2^{\epsilon n}$, where n is the number of attributes of Q. Thus the factor 2^{-n} in Theorem 4.18(2) cannot be replaced with anything greater than $2^{-(1-\epsilon)n}$.

Let $n \in \mathbb{N}$ be an integer, $0 < \epsilon < 1$ a fixed constant, and $A = \{a_1, \ldots, a_n\}$ a set of attributes with domain $\mathbb{N}$. Let $r := \lfloor \epsilon n / \log n \rfloor$. We assume that n is sufficiently large that $2^r > n$ holds. For every $B \in \binom{[n]}{r}$, let R_B be an r-ary relation with attributes B. Furthermore, for every $a \in A$, let R_a be a unary relation with the only attribute a. Let Q be the join of all these relations and let σ be the resulting schema.

For every $B \in \binom{[n]}{r}$, let $N_{R_B} = 2^r - 1$ and for every $a \in A$, let $N_{R_a} = 2$. Consider the linear program $L_Q(N_R : R \in \sigma)$. We obtain an optimal solution for this linear program by letting $x_{R_B} := n/\left(r\binom{n}{r}\right)$ and $x_{R_a} := 0$. To see that this is an optimum solution, observe that $y_a := \log(2^r - 1)/r$ is a feasible solution of the dual LP with the same cost.

We prove next that $\prod_R N_R^{x_R} = 2^n(1 - o(1))$:

$$\prod_{R\in\sigma} N_R^{x_R} = \left((2^r-1)^{n/\left(r\binom{n}{r}\right)}\right)^{\binom{n}{r}} \geq (2^r-1)^{n/r} \geq (2^r(1-1/n))^{n/r} = 2^n\,(1-1/n)^{n/r} = 2^n(1-o(1)).$$

The second inequality follows from $2^r > n$ and the last equality follows from the fact if n tends to infinity, then $(1 - 1/n)^n$ goes to $1/e$ and r goes to infinity as well.

To complete the example, we prove that $|Q(D)| \leq 2^{\epsilon n}$ for every instance D respecting the constraints N_R. Let D be a σ-instance with $|R(D)| = N_R$ for every $R \in \sigma$. From $N_{R_a} = 2$ it follows that in $Q(D)$ each attribute has at most two values, hence we can assume without loss of generality that $Q(D) \subseteq \{0, 1\}^n$. Thus each tuple in $t \in Q(D)$ can be viewed as a subset $A_t = \{a \in A : t(a) = 1\}$ of A. For every $B \in \binom{[n]}{r}$, it holds $\pi_B(Q(D)) \leq N_{R_B} = 2^r - 1$, hence the Vapnik-Chervonenkis dimension of $Q(D)$ is less than r. Thus by Sauer's Lemma, we have

$$|Q(D)| \leq n^r \leq n^{\epsilon n/\log n} = 2^{\epsilon n},$$

as claimed.

4.5.2 Hardness of better approximation

There is a gap of 2^n between the upper and lower bounds of Theorem 4.18, which means that both bounds approximate the maximum size of $|Q(D)|$ within a factor of 2^n. However, if $|Q(D)|$ is $2^{O(n)}$, then such an approximation is useless. We show that it is not possible to find a better approximation in polynomial time: the gap between an upper and a lower bound cannot be reduced to $2^{O(n^{1-\epsilon})}$ (under standard complexity-theoretic assumptions).

For the following statement, recall that ZPP is the class of decision problems that can be solved by a probabilistic polynomial-time algorithm with *zero-error*. What this means is that, on any input, the algorithm outputs the correct answer or "don't know", but the probability over the random choices of the algorithm that the answer is "don't know" is bounded by 1/2. Obviously P $\subseteq$ ZPP $\subseteq$ NP, and the assumption that ZPP $\neq$ NP is almost as believable as P $\neq$ NP (see [201]).

Theorem 4.20. *For a given query Q with schema σ and a given set of size constraints $(N_R : R \in \sigma)$, denote by M the maximum of $|Q(D)|$ over databases satisfying $|R(D)| = N_R$ for every $R \in \sigma$. If for some $\epsilon > 0$, there is a polynomial-time algorithm that, given a query Q with n attributes and size constraints N_R, computes two values M_L and M_U with $M_L \leq M \leq M_U$ and $M_U \leq M_L 2^{n^{1-\epsilon}}$, then* ZPP = NP.

For the proof of Theorem 4.20, we establish a connection between the query size and the maximum independent set problem (Lemma 4.22). Then we get our inapproximability result by reduction from the following result by Håstad:

Theorem 4.21 (Håstad [136])**.** *If for some $\epsilon_0 > 0$ there is a polynomial-time algorithm that, given an n-vertex graph G, can distinguish between the cases $\alpha(G) \leq n^{\epsilon_0}$ and $\alpha(G) \geq n^{1-\epsilon_0}$, then* ZPP = NP.

Following is the announced connection between worst-case query-size subject to relation-size constraints and maximum independent sets:

Lemma 4.22. *Let Q be a join query with schema σ and let $N_R := 2$ for all $R \in \sigma$. Let G be the primal graph of Q and let $\alpha(G)$ be the size of the maximum independent set in G. The maximum of $|Q(D)|$, taken over database instances satisfying $|R(D)| = N_R$ for every $R \in \sigma$, is exactly $2^{\alpha(G)}$.*

Proof. Let A_R be the attributes of $R \in \sigma$. For this proof we write A instead of A_σ. First we give a database D with $|Q(D)| \geq 2^{\alpha(G)}$. Let $I \subseteq A$ be an independent set of size $\alpha(G)$. Since I is independent, $|A_R \cap I|$ is either 0 or 1 for every $R \in \sigma$. If $|A_R \cap I| = 0$, then we define $R(D)$ to contain a tuple that is 0 on every attribute. If $A_R \cap I = \{a\}$, then we define $R(D)$ to contain a tuple that is 0 on every attribute and a tuple that is 1 on a and 0 on every attribute in $A_R \setminus \{a\}$. We claim that

$$Q(D) = \{t \in \mathrm{tup}(A) : t(a) \in \{0,1\} \text{ for all } a \in I, t(a) = 0 \text{ for all } a \in A \setminus I\}.$$

Clearly, the value of an attribute in I is either 0 or 1, and every attribute in $A \setminus I$ is forced to 0. Furthermore, any combination of 0 and 1 on the attributes of I is allowed as long as all the other attributes are 0. Thus $|Q(D)| = 2^{\alpha(G)}$. Note that a relation R with $|A_R \cap I| = 0$ contains only one tuple in the definition above. To satisfy the requirement $|R(D)| = N_R = 2$, we can add an arbitrary tuple to each such relation R; this cannot decrease $|Q(D)|$.

Next we show that if $|R(D)| = 2$ for every relation $R \in \sigma$, then $|Q(D)| \leq 2^{\alpha(G)}$. Since $|R(D)| = 2$ for every relation, every attribute in A can have at most two values in $Q(D)$; without loss of generality it can be assumed that $Q(D) \subseteq \{0,1\}^{|A|}$. Furthermore, it can be assumed (by a mapping of the domain of the attributes) that the all-0 tuple is in $Q(D)$.

Let S be the set of those attributes that have two values in $Q(D)$, i.e.,

$$S = \{a \in A : |\pi_{\{a\}}(Q(D))| = 2\}.$$

For every $a \in S$, let S_a be the set of those attributes that are the same as a in every tuple of $Q(D)$, i.e.,

$$S_a = \{b \in S : t(a) = t(b) \text{ for every } t \in Q(D)\}.$$

We define a sequence $a_1, a_2, \ldots$ of attributes by letting a_i be an arbitrary attribute in $S \setminus \bigcup_{j<i} S_{a_j}$. Let a_t be the last element in this sequence, which means that $\bigcup_{i=1}^{t} S_{a_i} = S$. We claim that $a_1, \ldots, a_t$ are independent in G, implying $t \leq \alpha(G)$. Assume that a_i and a_j $(i < j)$ are adjacent in G; this means that there is an $R \in \sigma$ with $a_i, a_j \in A_R$. By assumption, the all-0 tuple is in $R(D)$. As $a_i, a_j \in S$, there has to be a $t_1 \in R(D)$ with $t_1(a_i) = 1$ and a $t_2 \in R(D)$ with $t_2(a_j) = 1$. Since $|R(D)| = 2$ and the all-0 tuple is in $R(D)$, we have $t_1 = t_2$. But this means that a_i and a_j have the same value in both tuples in $R(D)$, implying $a_j \in S_{a_i}$. However, this contradicts the way the sequence was defined.

Now it is easy to see that $|Q(D)| \leq 2^t \leq 2^{\alpha(G)}$: by setting the value of $a_1, \ldots, a_t$, the value of every attribute in S is uniquely determined and the attributes in $A \setminus S$ are the same in every tuple of $Q(D)$. □

Proof of Theorem 4.20. We show that if such M_L and M_U could be determined in polynomial time, then we would be able to distinguish between the two cases of Theorem 4.21. Given an n-vertex graph $G = (V, E)$, we construct a query Q with attributes V and schema $\sigma = E$. For each edge $uv \in E$, there is a relation R_{uv} with attributes $\{u, v\}$. We set $N_R = 2$ for every relation $R \in \sigma$. Observe that the primal graph of Q is G. Thus by Lemma 4.22, $M = 2^{\alpha(G)}$.

Set $\epsilon_0 := \epsilon/2$. In case (1) of Theorem 4.21, $\alpha(G) \leq n^{\epsilon_0}$, hence $M_L \leq M \leq 2^{n^{\epsilon_0}}$ and

$$M_U \leq M_L 2^{n^{1-\epsilon}} \leq 2^{n^{\epsilon_0} + n^{1-\epsilon}} < 2^{n^{1-\epsilon_0}}$$

(if n is sufficiently large). On the other hand, in case (2) we have $\alpha(G) \geq n^{1-\epsilon_0}$, which implies $M_U \geq M = 2^{\alpha(G)} \geq 2^{n^{1-\epsilon_0}}$. Thus we can distinguish between the two cases by comparing M_U with $2^{n^{1-\epsilon_0}}$. □

CHAPTER 5

Fractional hypertree width

The notion of hypertree width, introduced by Gottlob et al. [116], gives wide classes of hypergraph properties that have unbounded treewidth, but still guarantee polynomial-time solvability of CSP instances. In Chapter 4, we have seen that bounded fractional edge cover number is another property ensuring polynomial-time solvability, and it is orthogonal to bounded hypertree width (see Figure 1.1 on page 16). In this chapter, we start a more systematic investigation of the interaction between fractional covers and hypertree width. We propose a new hypergraph invariant, the *fractional hypertree width*, which generalizes both the hypertree width and fractional edge cover number in a natural way.

Fractional hypertree width is an interesting hybrid of the "continuous" fractional edge cover number and the "discrete" hypertree width. We show that it has properties that are similar to the nice properties of hypertree width. In particular, we give an approximative game characterization of fractional hypertree width similar to the characterization of treewidth by the "cops and robber" game [217]. Furthermore, we prove that for classes $\mathcal{H}$ of bounded fractional hypertree width, the problem CSP($\mathcal{H}$) can be solved in polynomial time provided that a fractional hypertree decomposition of the underlying hypergraph is given together with the input instance. Unfortunately, we cannot expect that, for every fixed k, there is a polynomial-time algorithm for finding a fractional hypertree decomposition of width k: it is known that the problem is NP-hard even for $k = 2$ [100]. However, we show that we can find an approximate decomposition whose width is bounded by a (cubic) function of the fractional hypertree width. This is sufficient to show that CSP($\mathcal{H}$) is polynomial-time solvable for classes $\mathcal{H}$ of bounded fractional hypertree width, even if no decomposition is given in the input. Therefore, bounded fractional hypertree width is the so far most general hypergraph property that makes CSP($\mathcal{H}$) polynomial-time solvable. Note that this property is strictly more general than bounded hypertree width and bounded fractional edge cover number.

Publications. This chapter is based on two publications. Section 5.1 is based on the second half of an articled that appeared in *ACM Transactions on Algorithms* [126] (an extended abstract appeared in the proceedings of the SODA 2006 conference [124]). Section 5.2 is based on a single-author publication that appeared in *ACM Transactions on Algorithms* [184] (an extended abstract appeared in the proceedings of the SODA 2009 conference [183]).

5.1 Fractional hypertree decompositions

Let H be a hypergraph. A *generalized hypertree decomposition of* H [116] is a triple $(T, (B_t)_{t\in V(T)}, (C_t)_{t\in V(T)})$, where $(T, (B_t)_{t\in V(T)})$ is a tree decomposition of H and $(C_t)_{t\in V(T)}$ is a family of subsets of $E(H)$ such that for every $t \in V(T)$ we have $B_t \subseteq \bigcup C_t$. Here $\bigcup C_t$ denotes the union of the sets (hyperedges) in C_t, that is, the set $\{v \in V(H) \mid \exists e \in C_t : v \in e\}$. We call the sets B_t the *bags* of the decomposition and the sets C_t the *guards*. The *width* of $(T, (B_t)_{t\in V(T)}, (C_t)_{t\in V(T)})$ is $\max\{|C_t| \mid t \in V(T)\}$. The *generalized hypertree width* $\mathrm{ghw}(H)$ of H is the minimum of the widths of the generalized hypertree decompositions of H. The *edge cover number* $\rho(H)$ of a hypergraph is the minimum number of edges needed to cover all vertices; it is easy to see that $\rho(H) \geq \rho^*(H)$. Observe that the size of C_t has to be at least $\rho(H[B_t])$ and, conversely, for a given B_t there is always a suitable guard C_t of size $\rho(H[B_t])$. Therefore, $\mathrm{ghw}(H) \leq r$ if and only if there is a tree decomposition where $\rho(H[B_t]) \leq r$ for every $t \in V(T)$.

For the sake of completeness, let us mention that a *hypertree decomposition* of H is a generalized hypertree decomposition $(T, (B_t)_{t\in V(T)}, (C_t)_{t\in V(T)})$ that satisfies the following additional *special condition*: $(\bigcup C_t) \cap \bigcup_{u\in V(T_t)} B_u \subseteq B_t$ for all $t \in V(T)$. Recall that T_t denotes the subtree of the T with root t. The *hypertree width* $\mathrm{hw}(H)$ of H is the minimum of the widths of all hypertree decompositions of H. It has been proved in [8] that $\mathrm{ghw}(H) \leq \mathrm{hw}(H) \leq 3 \cdot \mathrm{ghw}(H) + 1$. This means that for our purposes, hypertree width and generalized hypertree width are equivalent. For simplicity, we will only work with generalized hypertree width.

Observe that for every hypergraph H without isolated vertices, we have $\mathrm{ghw}(H) \leq \mathrm{tw}(H) + 1$. Furthermore, if H is a hypergraph with $V(H) \in E(H)$ we have $\mathrm{ghw}(H) = 1$ and $\mathrm{tw}(H) = |V(H)| - 1$.

We now give an approximate characterization of (generalized) hypertree width by a game that is a variant of the *cops and robber* game [217], which characterizes treewidth: In the *robber and marshals game on* H [117], a robber plays against k marshals. The marshals move on the hyperedges of H, trying to catch the robber. Intuitively, the marshals occupy all vertices of the hyperedges where they are located. In each move, some of the marshals fly in helicopters to new hyperedges. The robber moves on the vertices of H. She sees where the marshals will be landing and quickly tries to escape, running arbitrarily fast along paths of H, not being allowed to run through a vertex that is occupied by a marshal before *and* after the flight (possibly by two different marshals). The marshals' objective is to land a marshal via helicopter on a hyperedge containing the vertex occupied by the robber. The robber tries to elude capture. The *marshal width* $\mathrm{mw}(H)$ of a hypergraph H is the least number k of marshals that have a winning strategy in the robber and marshals game played on H (see [6] or [117] for a formal definition).

It is easy to see that $\mathrm{mw}(H) \leq \mathrm{ghw}(H)$ for every hypergraph H. To win the game on a hypertree of generalized hypertree width k, the marshals always occupy guards of a decomposition and eventually capture the robber at a leaf of the tree. Conversely, it can be proved that $\mathrm{ghw}(H) \leq 3 \cdot \mathrm{mw}(H) + 1$ [8].

Observe that for every hypergraph H, the generalized hypertree width $\mathrm{ghw}(H)$ is less than or equal to the edge cover number $\rho(H)$: hypergraph H has a generalized hypertree decomposition consisting of a single bag containing all vertices and having a guard of size $\rho(H)$. On the other hand, the following two examples show that hypertree width and *fractional* edge cover number are incomparable.

Example 5.1. Consider the class of all graphs that only have disjoint edges. The treewidth and hypertree width of this class is 1, whereas the fractional edge cover number is unbounded.

Example 5.2. For $n \geq 1$, let H_n be the following hypergraph: H_n has a vertex v_S for every subset S of $\{1, \ldots, 2n\}$ of cardinality n. Furthermore, for every $i \in \{1, \ldots, 2n\}$ the hypergraph H_n has a hyperedge $e_i = \{v_S \mid i \in S\}$.

Observe that the fractional edge cover number $\rho^*(H_n)$ is at most 2, because the mapping x that assigns $1/n$ to every hyperedge e_i is a fractional edge cover of weight 2. Actually, it is easy to see that $\rho^*(H_n) = 2$.

We claim that the hypertree width of H_n is n. We show that H_n has a hypertree decomposition of width n. Let $S_1 = \{1, \ldots, n\}$ and $S_2 = \{n+1, \ldots, 2n\}$. We construct a generalized hypertree decomposition for H_n with a tree T having two nodes t_1 and t_2. For $i = 1, 2$, we let B_{t_1} contain a vertex V_S if and only if $S \cap S_i \neq \emptyset$. For each edge $e_j \in E(H_n)$, there is a bag of the decomposition that contains e_j: if $j \in S_i$, then B_{t_i} contains every vertex of e_j. We set the guard C_{t_i} to contain every e_j with $j \in S_i$. It is clear that $|C_{t_i}| = n$ and C_{t_i} covers B_{t_i}: vertex v_S is in B_{t_i} only if there is a $j \in S \cap S_i$, in which case $e_j \in C_{t_i}$ covers v_S. Thus this is indeed a generalized hypertree decomposition of width n for H_n and $\mathrm{ghw}(H_n) \leq n$ follows.

To see that $\mathrm{ghw}(H_n) > n - 1$, we argue that the robber has a winning strategy against $(n-1)$ marshals in the robber and marshals game. Consider a position of the game where the marshals occupy edges $e_{j_1}, \ldots, e_{j_{n-1}}$ and the robber occupies a vertex v_S for a set S with $S \cap \{j_1, \ldots, j_{n-1}\} = \emptyset$. Suppose that in the next round of the game the marshals move to the edges $e_{k_1}, \ldots, e_{k_{n-1}}$. Let $i \in S \setminus \{k_1, \ldots, k_{n-1}\}$. The robber moves along the edge e_i to a vertex v_R for a set $R \subseteq \{1, \ldots, 2n\} \setminus \{k_1, \ldots, k_{n-1}\}$ of cardinality n that contains i. If she plays this way, she can never be captured.

For a hypergraph H and a mapping $\gamma : E(H) \to [0, \infty)$, we let

$$B(\gamma) = \{v \in V(H) \mid \sum_{e \in E(H), v \in e} \gamma(e) \geq 1\}.$$

We may think of $B(\gamma)$ as the set of all vertices "blocked" by γ. Furthermore, we let $\mathrm{weight}(\gamma) = \sum_{e \in E} \gamma(e)$.

Definition 5.3. Let H be a hypergraph. A *fractional hypertree decomposition of* H is a triple $(T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$, where $(T, (B_t)_{t \in V(T)})$ is a tree decomposition of H and $(\gamma_t)_{t \in V(T)}$ is a family of mappings from $E(H)$ to $[0, \infty)$ such that for every $t \in V(T)$ we have $B_t \subseteq B(\gamma_t)$.

We call the sets B_t the *bags* of the decomposition and the mappings γ_t the *(fractional) guards*.

The *width* of $(T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ is $\max\{\mathrm{weight}(\gamma_t) \mid t \in V(T)\}$. The *fractional hypertree width* $\mathrm{fhw}(H)$ of H is the minimum of the widths of the fractional hypertree decompositions of H. Equivalently, $\mathrm{fhw}(H) \leq r$ if H has a tree decomposition where $\rho^*(B_t) \leq r$ for every bag B_t.

It is easy to see that the minimum of the widths of all fractional hypertree decompositions of a hypergraph H always exists and is rational. This follows from the fact that, up to an obvious equivalence, there are only finitely many tree decompositions of a hypergraph.

Clearly, for every hypergraph H we have

$$\mathrm{fhw}(H) \leq \rho^*(H) \quad \text{and} \quad \mathrm{fhw}(H) \leq \mathrm{ghw}(H).$$

Examples 5.1 and 5.2 above show that there are families of hypergraphs of bounded fractional hypertree width, but unbounded fractional edge cover number and unbounded generalized hypertree width.

It is also worth pointing out that for every hypergraph H,

$$\mathrm{fhw}(H) = 1 \iff \mathrm{ghw}(H) = 1.$$

To see this, note that if $\gamma : E(H) \to [0, \infty)$ is a mapping with $\mathrm{weight}(\gamma) = 1$ and $B \subseteq B(\gamma)$, then $B \subseteq e$ for all $e \in E(H)$ with $\gamma(e) > 0$. Thus instead of using γ as a guard in a fractional hypertree decomposition, we may use the integral guard $\{e\}$ for any $e \in E(H)$ with $\gamma(e) > 0$. Let us remark that $\mathrm{ghw}(H) = 1$ if and only if H is acyclic [116].

5.1.1 Finding decompositions

For the algorithmic applications, it is essential to have algorithms that find fractional hypertree decompositions of small width. The question is whether for any fixed $r > 1$ there is a polynomial-time algorithm that, given a hypergraph H with $\mathrm{fhw}(H) \leq r$, computes a fractional hypertree decomposition of H of width at most r. Unfortunately, a very recent result of Fischl et al. [100] shows that this problem is NP-hard even for $r = 2$ (earlier, it was known that the problem is NP-hard if r is part of the input [105, 184]).

Given the hardness of finding the best possible decomposition, one has to be satisfied with approximate solutions. In Section 5.2, we present an algorithm that approximates fractional hypertree width in the following sense:

Theorem 5.4. *For every $r \geq 1$, there is an $n^{O(r^3)}$ time algorithm that, given a hypergraph with fractional hypertree width at most r, finds a fractional hypertree decomposition of width $O(r^3)$.*

The main technical challenge in the proof of Theorem 5.4 is finding a separator with bounded fractional edge cover number that separates two sets X, Y of vertices. An approximation algorithm is given in Section 5.2.1 for this problem, which finds a separator of weight $O(r^3)$ if a separator of weight r exists. This algorithm is used to find balanced separators, which in turn is used to construct a tree decomposition (Section 5.2.2).

5.1.2 Algorithmic applications

In this section, we discuss how problems can be solved by fractional hypertree decompositions of bounded width. First we give a basic result, which formulates why tree decompositions and width measures are useful in the algorithmic context: CSP can be efficiently solved if we can polynomially bound the number of solutions in the bags. Recall that H_I denotes the hypergraph of a CSP instance I. If $(T, (B_t)_{t \in V(T)})$ is a tree decomposition of H_I, then $I[B_t]$ denotes the instance induced by bag B_t, see Definition 4.5.

Lemma 5.5. *There is an algorithm that, given a CSP instance, a hypertree decomposition $(T, (B_t)_{t \in V(T)})$ of H_I, and for every $t \in V(t)$ a list L_t of all solutions of $I[B_t]$, decides in time $C \cdot \|I\|^{O(1)}$ if I is satisfiable (and computes a solution if it is), where $C := \max_{t \in V(T)} |L_t|$.*

Proof. Define $V_t := \bigcup_{t \in V(T_t)} B_t$. For each $t \in V(T)$, our algorithm constructs the list $L'_t \subseteq L_t$ of those solutions of $I[B_t]$ that can be extended to a solution of $I[V_t]$. Clearly, I has a solution if and only if L'_{t_0} is not empty for the root t_0 of the tree decomposition.

The algorithm proceeds in a bottom-up manner: when constructing the list L'_t, we assume that for every child t' of t, the lists $L'_{t'}$ are already available. If t is a leaf node, then $V_t = B_t$, and $L'_t = L_t$. Assume now that t has children $t_1, \ldots, t_k$. We claim that a solution α of $I[B_t]$ can be extended to $I[V_t]$ if and only if for each $1 \leq i \leq k$, there is a solution α_i of $I[V_{t_i}]$ that is compatible with α (that is, α and α_i assign the same values to the variables in $B_t \cap V_{t_i} = B_t \cap B_{t_i}$). The necessity of this condition is clear: the restriction of a solution of $I[V_t]$ to V_{t_i} is clearly a solution of $I[V_{t_i}]$. For sufficiency, suppose that the solutions α_i exist for every child t_i. They can be combined to an assignment α' on V_t extending α in a well-defined way: every variable $v \in V_{t_i} \cap V_{t_j}$ is in B_t, thus $\alpha_i(v) = \alpha_j(v) = \alpha(v)$ follows for such a variable. Now α' is a solution of $I[V_t]$: for each constraint of $I[V_t]$, the variables of the constraint are contained either in B_t or in V_{t_i} for some $1 \leq i \leq k$, thus α or α_i satisfies the constraint, implying that α' satisfies it as well.

Therefore, L'_t can be determined by first enumerating every solution $\alpha \in L_t$, and then for each i, checking whether L'_{t_i} contains an assignment α_i compatible with α. This check can be efficiently

performed the following way. Recall that solutions α and α_i are compatible if their restriction to $B_t \cap B_{t_i}$ is the same assignment. Therefore, after computing L'_{t_i}, we restrict every $\alpha_i \in L'_{t_i}$ to $B_t \cap B_{t_i}$ and store these restrictions in a trie data structure for easy membership tests. Then to check if there is an $\alpha_i \in L'_{t_i}$ compatible with α, all we need to do is to check if the restriction of α to $B_t \cap B_{t_i}$ is in the trie corresponding to L'_{t_i}, which can be checked in $\|I\|^{O(1)}$. Thus L'_t (and the corresponding trie structure) can be computed in time $|L_t| \cdot \|I\|^{O(1)}$. As every other part of the algorithm can be done in time $\|I\|^{O(1)}$, it follows that the total running time can be bounded by $C \cdot \|I\|^{O(1)}$. Using standard bookkeeping techniques, it is not difficult to extend the algorithm such that it actually returns a solution if one exists. □

Lemma 5.5 tells us that if we have a tree decomposition where we can give a polynomial bound on the number of solutions in the bags for some reason (and we can enumerate all these solutions), then the problem can be solved in polynomial time. Observe that in a fractional hypertree decomposition every bag has bounded fractional edge cover number and hence Theorem 4.6 can be used to enumerate all the solutions. It follows that if a fractional hypertree decomposition of bounded width is given in the input, then the problem can be solved in polynomial time.

Theorem 5.6. *There is an algorithm that, given a CSP instance, a fractional hypertree decomposition $(T, (B_t)_{t \in V(T)}, \gamma)$ of H_I having width at most r, decides in time $\|I\|^{r+O(1)}$ if I is satisfiable (and computes a solution if it is).*

Moreover, if we know that the hypergraph has fractional hypertree width at most r (but no decomposition is given in the input), then we can use brute force to find a fractional hypertree decomposition of width at most r by trying every possible decomposition and then solve the problem in polynomial time. This way, the running time is polynomial in the input size times an (exponential) function of the number of variables. This immediately shows that if we restrict CSP to a class of hypergraphs whose fractional hypertree width is at most a constant r, then the problem is fixed-parameter tractable parameterized by the number of variables. To get rid of the exponential factor depending on the number of variables and obtain a polynomial-time algorithm, we can replace the brute force search for the decomposition by the approximation algorithm of Theorem 5.4.

Theorem 5.7. *Let $r \geq 1$. Then there is a polynomial-time algorithm that, given a CSP instance I of fractional hypertree width at most r, decides if I is satisfiable (and computes a solution if it is).*

Proof. Let I be a CSP instance of fractional hypertree width at most r, and let $(T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ be the fractional hypertree decomposition of $\mathcal{H}_I$ of width $O(r^3)$ computed by the algorithm of Theorem 5.4. By the definition, the hypergraph of $I[B_t]$ has fractional edge cover number $O(r^3)$ for every bag B_t. Thus by Theorem 4.6, the list L_t of the solutions of $I[B_t]$ has size at most $\|I\|^{O(r^3)}$ and can be determined in time $\|I\|^{O(r^3)}$. Therefore, we can find a solution in time $\|I\|^{O(r^3)}$ using the algorithm of Lemma 5.5. □

As a corollary of Theorem 5.7, we obtain that whenever $\mathcal{H}$ has bounded fractional hypertree width, then CSP($\mathcal{H}$) is polynomial-time solvable. This makes "bounded fractional hypertree width" the strictly more general known hypergraph property that makes the CSP polynomial-time solvable.

Corollary 5.8. *If $\mathcal{H}$ has bounded fractional hypertree width, then* CSP($\mathcal{H}$) *can be solved in polynomial time.*

5.1.3 The robber and army game

As robbers are getting ever more clever, it takes more and more powerful security forces to capture them. In the *robber and army game on a hypergraph* H, a robber plays against a general commanding an army of r battalions of soldiers. The general may distribute his soldiers arbitrarily on the hyperedges. However, a vertex of the hypergraph is only blocked if the number of soldiers on all hyperedges that contain this vertex adds up to the strength of at least one battalion. The game is then played like the robber and marshals game.

Definition 5.9. Let H be a hypergraph and r a nonnegative real. The *robber and army game on H with r battalions* (denoted by $\mathrm{RA}(H,r)$) is played by two players, the *robber* and the *general*. A *position* of the game is a pair (γ, v), where $v \in V(H)$ and $\gamma : E(H) \to [0,\infty)$ with $\mathrm{weight}(\gamma) \leq r$. To start a game, the robber picks an arbitrary v_0, and the initial position is $(0, v_0)$, where 0 denote the constant zero mapping.

In each round, the players move from the current position (γ, v) to a new position (γ', v') as follows: The general selects γ', and then the robber selects v' such that there is a path from v to v' in the hypergraph $H \setminus \big(B(\gamma) \cap B(\gamma')\big)$.

If a position (γ, v) with $v \in B(\gamma)$ is reached, the play ends and the general wins. If the play continues forever, the robber wins.

The *army width* $\mathrm{aw}(H)$ of H is the least r such that the general has winning strategy for the game $\mathrm{RA}(H,r)$.

Again, it is easy to see that $\mathrm{aw}(H)$ is well-defined and rational (observe that two positions (γ_1, v) and (γ_2, v) are equivalent if $B(\gamma_1) = B(\gamma_2)$ holds).

Theorem 5.10. *For every hypergraph H,*

$$\mathrm{aw}(H) \leq \mathrm{fhw}(H) \leq 3 \cdot \mathrm{aw}(H) + 2.$$

The rest of this subsection is devoted to a proof of this theorem. The proof is similar to the proof of the corresponding result for the robber and marshal game and generalized hypertree width in [8], which in turn is based on ideas from [205, 217].

Let H be a hypergraph and $\gamma, \sigma : E(H) \to [0,\infty)$. For a set $W \subseteq V(H)$, we let

$$\mathrm{weight}(\gamma|W) = \sum_{\substack{e \in E(H) \\ e \cap W \neq \emptyset}} \gamma(e).$$

A mapping $\sigma : E(H) \to [0,\infty)$ is a *balanced separator for* γ if for every connected component R of $H \setminus B(\sigma)$,

$$\mathrm{weight}(\gamma|R) \leq \frac{\mathrm{weight}(\gamma)}{2}.$$

Lemma 5.11. *Let H be a hypergraph with $\mathrm{aw}(H) \leq r$ for some nonnegative real r. Then every $\gamma : E(H) \to [0,\infty)$ has a balanced separator of weight r.*

Proof. Suppose for contradiction that $\gamma : E(H) \to [0,\infty)$ has no balanced separator of weight r. We claim that the robber has a winning strategy for the game $\mathrm{RA}(H,r)$. The robber simply maintains the invariant that in every position (σ, v) of the game, v is contained in the connected component R of $H \setminus B(\sigma)$ with $\mathrm{weight}(\gamma|R) > \mathrm{weight}(\gamma)/2$.

To see that this is possible, let (σ, v) be such a position. Suppose that the general moves from σ to σ', and let R' be the connected component of $H \setminus B(\sigma')$ with $\text{weight}(\gamma|R') > \text{weight}(\gamma)/2$. Then there must be some $e \in E(H)$ such that $e \cap R \neq \emptyset$ and $e \cap R' \neq \emptyset$, because otherwise we had

$$\text{weight}(\gamma) = \text{weight}(\gamma)/2 + \text{weight}(\gamma)/2 < \text{weight}(\gamma|R) + \text{weight}(\gamma|R') \leq \text{weight}(\gamma),$$

which is impossible. Thus the robber can move from R to R' via the edge e. □

Let H be a hypergraph and H' an induced subhypergraph of H. Then the *restriction* of a mapping $\gamma : E(H) \to [0, \infty)$ to H' is the mapping $\gamma' : E(H') \to [0, \infty)$ defined by

$$\gamma'(e') = \sum_{\substack{e \in E(H) \\ e \cap V(H') = e'}} \gamma(e).$$

Note that $\text{weight}(\gamma') \leq \text{weight}(\gamma)$ and $B(\gamma') = B(\gamma) \cap V(H')$. The inequality may be strict because edges with nonempty weight may have an empty intersection with $V(H')$. Conversely, the *canonical extension* of a mapping $\gamma' : E(H') \to [0, \infty)$ to H is the mapping $\gamma : E(H) \to [0, \infty)$ defined by

$$\gamma(e) = \frac{\gamma'(e \cap V(H'))}{|\{e_1 \in E(H) \mid e_1 \cap V(H') = e \cap V(H)\}|}$$

if $e \cap V(H') \neq \emptyset$ and $\gamma(e) = 0$ otherwise. Intuitively, for every $e' \in E(H')$, we distribute the weight of e' equally among all the edges $e \in E(H)$ whose intersection with $V(H')$ is exactly e'. Note that $\text{weight}(\gamma) = \text{weight}(\gamma')$ and $B(\gamma') = B(\gamma) \cap V(H')$.

Proof (of Theorem 5.10). Let H be a hypergraph. To prove that $\text{aw}(H) \leq \text{fhw}(H)$, let $(T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ be a fractional hypertree decomposition of H having width $\text{fhw}(H)$. We claim that the general has a winning strategy for $\text{RA}(H, r)$. Let $(0, v_0)$ be the initial position. The general plays in such a way that all subsequent positions are of the form (γ_t, v) such that $v \in B_u$ for some $u \in V(T_t)$. Intuitively, this means that the robber is trapped in the subtree below t. Furthermore, in each move the general reduces the height of t. He starts by selecting γ_{t_0} for the root t_0 of T. Suppose the game is in a position (γ_t, v) such that $v \in B_u$ for some $u \in V(T_t)$. If $u = t$, then the robber has lost the game. So let us assume that $u \neq t$. Then there is a child t' of t such that $u \in V(T_{t'})$. The general moves to $\gamma_{t'}$. Suppose the robber escapes to a v' that is not contained in $B_{u'}$ for any $u' \in T_{t'}$. Then there is a path from v to v' in $H \setminus (B(\gamma_t) \cap B(\gamma_{t'}))$ and hence in $H \setminus (B_t \cap B_{t'})$. However, it follows easily from the fact that $(T, (B_t)_{t \in T})$ is a tree decomposition of H that every path from a bag in $T_{t'}$ to a bag in $T \setminus T_{t'}$ must intersect $B_t \cap B_{t'}$. This proves that $\text{aw}(H) \leq \text{fhw}(H)$.

For the second inequality, we shall prove the following stronger claim:

Claim: Let H be a hypergraph with $\text{aw}(H) \leq r$ for some nonnegative real r. Furthermore, let $\gamma : E(H) \to [0, \infty)$ such that $\text{weight}(\gamma) \leq 2r + 2$. Then there exists a fractional hypertree decomposition of H of width at most $3r + 2$ such that $B(\gamma)$ is contained in the bag of the root of this decomposition.

Note that for $\gamma = 0$, the claim yields the desired fractional hypertree decomposition of H.

Proof of the claim: The proof is by induction on the cardinality of $V(H) \setminus B(\gamma)$.

By Lemma 5.11, there is a balanced separator of weight at most r for γ in H. Let σ be such a separator, and define $\chi : E(H) \to [0, \infty)$ by $\chi(e) = \gamma(e) + \sigma(e)$. Then $\text{weight}(\chi) \leq 3r + 2$, and $B(\gamma) \cup B(\sigma) \subseteq B(\chi)$.

If $V(H) = B(\chi)$ (this is the induction basis), then the 1-node decomposition with bag $V(H)$ and guard χ is a fractional hypertree decomposition of H of width at most $3r + 2$.

Otherwise, let $R_1, \ldots, R_m$ be the connected components of $H \setminus B(\chi)$. Note that we cannot exclude the case $m = 1$ and $R_1 = V(H) \setminus B(\chi)$.

For $1 \leq i \leq m$, let e_i be an edge of H such that $e_i \cap R_i \neq \emptyset$, and let S_i be the unique connected component of $H \setminus B(\sigma)$ with $R_i \subseteq S_i$. Note that $\text{weight}(\gamma|S_i) \leq r + 1$, because σ is a balanced separator for γ. Let $\chi_i : E(H) \to [0, \infty)$ be defined by

$$\chi_i(e) = \begin{cases} 1 & \text{if } e = e_i, \\ \sigma(e) + \gamma(e) & \text{if } e \neq e_i \text{ and } S_i \cap e \neq \emptyset, \\ \sigma(e) & \text{otherwise.} \end{cases}$$

Then

$$\text{weight}(\chi_i) \leq 1 + \text{weight}(\sigma) + \text{weight}(\gamma|S_i) \leq 2r + 2$$

and $B(\chi_i) \setminus R_i \subseteq B(\chi)$ (as e_i cannot intersect any R_j with $i \neq j$). Let $H_i = H[R_i \cup B(\chi_i)]$ and observe that

$$V(H_i) \setminus B(\chi_i) \subseteq R_i \setminus e_i \subset R_i \subseteq V(H) \setminus B(\gamma)$$

(the first inclusion holds because $\chi_i(e_i) = 1$). Thus the induction hypothesis is applicable to H_i and the restriction of χ_i to H_i. It yields a fractional hypertree decomposition $(T^i, (B^i_t)_{t \in V(T^i)}, (\gamma^i_t)_{t \in V(T^i)})$ of H_i of weight at most $3r + 2$ such that $B(\chi_i)$ is contained in the bag $B^i_{t^i_0}$ of the root t^i_0 of T^i.

Let T be the disjoint union of $T^1, \ldots, T^m$ together with a new root t_0 that has edges to the roots t^i_0 of the T^i. Let $B_{t_0} = B(\chi)$ and $B_t = B^i_t$ for all $t \in V(T^i)$. Moreover, let $\gamma_{t_0} = \chi$, and let γ_t be the canonical extension of γ^i_t to H for all $t \in V(T^i)$.

It remains to prove that $(T, (B_t)_{t \in V(T)}, (\gamma_t)_{t \in V(T)})$ is a fractional hypertree decomposition of H of width at most $3r + 2$. Let us first verify that $(T, (B_t)_{t \in V(T)})$ is a tree decomposition.

- Let $v \in V(H)$. To see that $\{v \in V(T) \mid v \in B_t\}$ is connected in T, observe that $\{t \in V(T^i) \mid v \in B_{t_i}\}$ is connected (maybe empty) for all i. If $v \in R_i$ for some i, then $v \notin V(H_j) = R_j \cup B(\chi_j)$ for any $i \neq j$ (as R_i and R_j are disjoint and we have seen that $B(\chi_j) \setminus R_j \subseteq B(\chi)$) and this this already shows that $\{t \in V(T) \mid v \in B_t\}$ is connected. Otherwise, $v \in B(\chi_i) \setminus R_i \subseteq B(\chi) = B_{t_0}$ for all i such that $v \in V(H_i)$. Again this shows that $\{v \in V(T) \mid v \in B_t\}$ is connected.

- Let $e \in E(H)$. Either $e \subseteq B(\chi) = B_{t_0}$, or there is exactly one i such that $e \subseteq R_i \cup B(\chi_i)$. In the latter case, $e \subseteq B_t$ for some $t \in V(T^i)$.

It remains to prove that $B_t \subseteq B(\gamma_t)$ for all $t \in T$. For the root, we have $B_{t_0} = B(\gamma_{t_0})$. For $t \in V(T^i)$, we have $B_t \subseteq B(\gamma^i_t) = B(\gamma_t) \cap V(H_i) \subseteq B(\gamma_t)$. Finally, note that $\text{weight}(\gamma_t) \leq 3r + 2$ for all $t \in V(T)$. This completes the proof of the claim. □

5.2 Approximating fractional hypertree width

The main result of this section is an algorithm that computes approximately optimal fractional hypertree decompositions. More precisely, we show that for every fixed $w \geq 1$, there is a polynomial-time algorithm that, given a hypergraph H with fractional hypertree width at most w, computes a tree decomposition of H with fractional hypertree width $O(w^3)$ (Theorem 5.17).

Algorithms for finding tree decompositions and characterization theorems for (generalizations of) treewidth often follow a certain pattern. For example, the same high-level idea is used for treewidth [102, Section 11.2], rank width [198, 199], hypertree width [8], and branch width of matroids and submodular functions [200]. Simplifying somewhat, this general pattern can be summarized the following way: We decompose the problem into two parts by finding a small balanced separation, then a tree decomposition for each part is constructed using the algorithm recursively, and finally the tree decompositions for the parts are joined in an appropriate way to obtain a tree decomposition for the original problem. A balanced separation of a subset W is a partition (A, B) of W and a set S separating A and B, such that A and B are both "small" compared to W (the exact definition of small depends on the actual type of tree decomposition we are looking for). Depending on the approximation ratio and the running time we are trying to achieve, the problem of finding a balanced separation is either reduced to a sparsest cut problem or (using brute force) it is reduced to the problem of finding a small (A, B)-separator, i.e., a set whose deletion disconnects A and B.

Can we use a similar approach for constructing fractional hypertree decompositions? With appropriate modifications, the recursive algorithm works for such decompositions as well (Section 5.2.2). The crucial question is how to find a balanced separation where S has small fraction edge cover number. Using brute force in a not completely trivial way, the search for a balanced separation can be reduced to finding an (A, B)-separator with small fractional edge cover number (Lemma 5.16). The main technical contribution of the paper is an approximation algorithm for finding such separators: If there is an (A, B)-separator with fractional edge cover number at most w, then the algorithm finds an (A, B)-separator with fractional edge cover number $O(w^3)$ (Section 5.2.1). The running time is polynomial for every fixed w.

For other types of tree decompositions, the corresponding (A, B)-separation problem can be solved using flow techniques, brute force, or submodularity. None of these techniques seem to be relevant when the goal is to minimize the fractional edge cover number of the separator; we need completely different techniques. The main idea is the following. Suppose we are looking for an (A, B)-separator S with fractional edge cover number $w < 2$. As the fractional edge cover number is an upper bound on maximum independent set size, any two vertices in S are adjacent, i.e., S induces a clique. The structure of separating cliques is well understood: Every graph has a unique decomposition by clique separators [222]. Our algorithm for finding a separator with small fractional edge cover number can be thought of as a generalization of finding clique separators. A tempting way of generalizing this idea for larger w would be to suppose that every separator with fractional edge cover number at most w can be covered by $f(w)$ cliques for some function f. However, this is not true: it can be shown that we might need an unbounded number of cliques. Nevertheless, we manage to transform the instance in such a way that it can be assumed that the separator we are looking for can be covered by w cliques. Then we locate these cliques using a combination of brute force, clique separator decompositions, and linear programming.

5.2.1 Finding approximate separators

Let $A, B \subseteq V(H)$ be two sets of vertices. An (A, B)-*separator* is a set $S \subseteq V(H)$ such that there is no path connecting a vertex of $A \setminus S$ with a vertex of $B \setminus S$ in the hypergraph $H \setminus S$. In particular, such an S has to contain every vertex of $A \cap B$. The aim of this section is to give an approximation algorithm for the problem of finding an (A, B)-separator with minimum fractional edge cover number.

We say that two nonadjacent vertices u, v of H are w-*attached* for some $w \geq 1$ if $\rho^*_H(N(v) \cap N(u)) > w$ (here $N(v)$ is the set of neighbors of v, not including v itself). If u, v are w-attached and S is an (A, B)-separator with $\rho^*_H(S) \leq w$ covering neither u nor v, then u and v are in the same connected component of $H \setminus S$. This means that S remains an (A, B)-separator even if we add an edge between u and v. Thus adding edges between w-attached vertices does not change the problem significantly. More precisely, the following lemma shows that we can reduce the problem to a situation where nonadjacent vertices are not w-attached. This property of the hypergraph will play an important role in the algorithm.

Lemma 5.12. *Let H be a hypergraph, $A, B \subseteq V(H)$ sets of vertices, and $w \geq 1$ a rational number. We can construct in time polynomial in $\|H\|$ a hypergraph H^+ on the same set of vertices such that*

1. *If vertices u and v are not adjacent in H^+, then they are not w-attached.*
2. *If S is an (A, B)-separator in H with $\rho^*_H(S) \leq w$, then S is an (A, B)-separator in H^+ with $\rho^*_{H^+}(S) \leq w$.*
3. *If S is an (A, B)-separator in H^+, then S is an (A, B)-separator in H with $\rho^*_H(S) \leq 2\rho^*_{H^+}(S)$.*

Proof. We construct a sequence of hypergraphs. Let $H_0 = H$. Let (u, v) be an arbitrary pair of nonadjacent vertices that are w-attached in H_{i-1}. Hypergraph H_i is the same as H_{i-1} with an extra edge $\{u, v\}$. If there is no such pair (u, v) in H_{i-1}, then we stop the construction of the sequence. It is clear that the sequence has polynomial length (as at most $O(|V(H)|^2)$ new edges can be added) and constructing H_i from H_{i-1} can be done in polynomial time. Let $H^+ = H_k$ be the last hypergraph in the sequence. Statement 1 is immediate from the way the sequence is constructed.

To prove Statement 2, suppose that S is an (A, B)-separator in $H = H_0$. Since the edges of H are a subset of the edges of H^+, we have $\rho^*_{H^+}(S) \leq \rho^*_H(S) \leq w$. We prove by induction that S is an (A, B)-separator in every H_i. Suppose that this is true for H_{i-1}, but there is a path P from a vertex of A to a vertex of B in $H_i \setminus S$. Let $e_i = u_i v_i$ be the edge that was added to H_{i-1} to obtain H_i. If P does not use e_i, then P is also a path in H_{i-1}, contradicting the induction hypothesis that S is an (A, B)-separator in H_{i-1}. Thus $P = P_1 u_i v_i P_2$ for some subpaths P_1 and P_2 (by swapping u_i and v_i if necessary, we may assume that P reaches u_i before v_i). By the definition of e_i,

$$\rho^*_H(N(v_i) \cap N(u_i)) \geq \rho^*_{H_{i-1}}(N(v_i) \cap N(u_i)) > w \geq \rho^*_H(S),$$

which means that there is a vertex $q \in (N(v_i) \cap N(u_i)) \setminus S$. The walk $P_1 u_i q v_i P_2$ connects a vertex of A and a vertex of B in $H_{i-1} \setminus S$, contradicting the induction hypothesis.

To prove Statement 3, observe first that the edges of H are a subset of the edges of H^+, thus if S is an (A, B)-separator in H^+, then it is an (A, B)-separator in H as well. Consider a fractional edge cover γ of S in H^+ with weight$(\gamma) = w'$. Suppose that $\gamma(e) = x$ for an edge $e = \{u, v\}$ not present in H. In this case, we set the weight of this edge to 0, and increase by x the weight of two edges: an arbitrary edge $e_u \in E(H)$ that contains u and an arbitrary edge $e_v \in E(H)$ that contains v (such edges exist, since we assumed that there are no isolated vertices in the hypergraph). It is clear that the resulting weight assignment is also a fractional edge cover. We repeat this step until the weight assignment is 0 on every edge not present in H. It is easy to see that the weight of the assignment increases to at most $2w'$, thus $\rho^*_H(S) \leq 2\rho^*_{H^+}(S)$. □

The following result follows from the fact that a decomposition of a graph by separating cliques can be found in polynomial time [222, 230] (clique K is a separating clique of H if $H \setminus K$ has not connected). For the convenience of the reader, we give here a self-contained proof of the main idea in the form we use.

Lemma 5.13. *Given a graph G, it is possible to construct in time polynomial in $\|G\|$ a set $\mathcal{C}$ of at most $|V(G)|$ connected subsets such that*

1. *if K is a clique of G, then $K \subseteq C$ for some $C \in \mathcal{C}$, and*
2. *if K is a clique of G and $C \in \mathcal{C}$, then $C \setminus K$ is contained in a connected component of $G \setminus K$.*

Proof. We construct a sequence of graphs as follows. Let $G_0 = G$. Suppose that G_{i-1} has an induced cycle H of length at least 4; let v_i, u_i be two nonadjacent vertices of H. We define G_i to be the same as G_{i-1}, with an extra edge $e_i = v_i u_i$. If G_{i-1} has no such cycle H (i.e., G_{i-1} is a chordal graph), then we stop the construction of the sequence. Let G_k be the last graph in the sequence. Let $\mathcal{C}$ be the set of inclusionwise maximal cliques of G_k. It is well known that chordal graph G_k has at most $|V(G_k)| = |V(G)|$ maximal cliques (cf. [112]).

Every clique of G is a clique of G_k, thus Statement 1 is clear from the definition of $\mathcal{C}$. To prove Statement 2, for every $C \in \mathcal{C}$ and clique K of G, we show that $C \setminus K$ is contained in a connected component of $G_i \setminus K$ for every $1 \leq i \leq k$. This is clear for G_k, as C is a clique in G_k. Suppose that $C \setminus K$ is in a connected component of $G_i \setminus K$ but $a, b \in C \setminus K$ are in different connected components of $G_{i-1} \setminus K$. Let P be a path from a to b in $G_i \setminus K$. Path P has to go through the edge $e_i = u_i v_i$ used in the definition of G_i, otherwise it would be a path in $G_{i-1} \setminus K$ as well. Thus the path P can be written as $P = aP_1 u_i v_i P_2 b$ (assuming without loss of generality that P reaches u_i before v_i). There is an induced cycle H in G_{i-1} that contains u_i and v_i. Since $u_i, v_i \notin K$ and $H \setminus K$ is connected (as K is a clique and H is an induced cycle), there is a path R in $G_{i-1} \setminus K$ that connects u_i and v_i. Now $aP_1 u_i R v_i P_2 b$ is a walk from a to b in $G_{i-1} \setminus K$, a contradiction. □

For illustrative purposes, we show how Lemma 5.13 implies that all the minimal separating cliques can be enumerated in polynomial time (although we do not use this result here).

Corollary 5.14. *Given a graph G, it is possible to enumerate all the inclusionwise minimal separating cliques of G in time polynomial in $\|G\|$.*

Proof. Construct the sets $\mathcal{C}$ of Lemma 5.13 and consider the chordal graph G_k. We claim that every minimal separating clique of G is a minimal separating clique of G_k. Suppose that a clique K separates a and b in G, but there is a path P between a and b in $G_k \setminus K$. Each edge e of P is a clique of size 2 in G_k, hence the endpoints of e are contained in some set $C_i \in \mathcal{C}$. This means that the two endpoints are in the same connected component of $G \setminus K$ and it follows that every vertex of the path P (including a and b) are in the same component, a contradiction. Thus if K is a minimal separating clique in G, then it is a separating clique in G_k. Furthermore, as G_k is a supergraph of G, minimality of K in G implies its minimality in G_k as well. In a chordal graph, every minimal separating clique is the intersection of two maximal cliques. Thus all the minimal separating cliques can be enumerated by taking the intersection of every pair $C_i, C_j \in \mathcal{C}$ and checking whether it is really a minimal separating clique. □

Lemma 5.15. *Let H be a hypergraph, $A, B \subseteq V(H)$ two sets of vertices, and $w \geq 1$ a rational number. There is an algorithm that, in time $\|H\|^{O(w)}$, either*

- *correctly concludes that there is no (A, B)-separator S with $\rho^*_H(S) \leq w$, or*

- *produces an (A,B)-separator S' with $\rho^*_H(S') \leq w^3 + 4w$.*

Proof. The algorithm first constructs the hypergraph H^+ of Lemma 5.12 and then tries to find an (A,B)-separator in H^+. By Lemma 5.12(2), if H has an (A,B)-separator S with $\rho^*_H(S) \leq w$, then S is an (A,B)-separator in H^+ as well and $\rho^*_{H^+}(S) \leq w$. In this case, our algorithm detailed below will be able to find an (A,B)-separator S' in H^+ with $\rho^*_{H^+}(S') \leq w^3/2 + 2w$. By Lemma 5.12(3), such an S' is an (A,B)-separator in H with $\rho^*_H(S') \leq w^3 + 4w$.

Suppose that there is an (A,B)-separator S in H^+ with $\rho^*_{H^+}(S) \leq w$. In the rest of the proof, we show how to find the required separator S' if we know a maximum independent set I_S of S. Since the fractional edge cover number of S is at most w, the size of I_S is also at most w. Thus trying all possible sets I_S adds a factor of $\|H^+\|^{O(w)} = \|H\|^{O(w)}$ to the running time.

Denote by $N(v)$ the neighbors of vertex v in H^+. Suppose that $I_S = \{v_1, \ldots, v_k\}$ (for some $k \leq w$) is a maximum independent set of S. By the definition of H^+, we have $\rho^*_{H^+}(N(v_i) \cap N(v_j)) \leq w$ for every $1 \leq i < j \leq k$. Thus $X = \bigcup_{1 \leq i < j \leq k}(N(v_i) \cap N(v_j))$ has fractional edge cover number at most $\binom{k}{2} w \leq w^3/2$. In the rest of the algorithm, we try to find a set Y with $\rho^*_{H^+}(Y) \leq 2w$ such that $S' := X \cup Y$ is an (A,B)-separator in H^+.

Let $N_i = (N(v_i) \cup \{v_i\}) \setminus X$ for $i = 1, \ldots, k$. Let us note first that $N_i \cap N_j = \emptyset$ if $i \neq j$: Vertices v_i and v_j are not adjacent and every vertex of $N(v_i) \cap N(v_j)$ is in X. Since $v_1, \ldots, v_k$ is a maximum independent set of S, each vertex of $S \setminus X$ is in one of the N_i's. Observe that $N_i \cap S$ is not empty, since it contains v_i (here we use that v_i cannot be in X, since it is not adjacent to any other v_j). Furthermore, for every $1 \leq i \leq k$, $N_i \cap S$ is a clique of N_i (this is a crucial point of the proof). To see this, suppose that $v'_i, v''_i \in N_i \cap S$ are nonadjacent vertices; clearly, it is not possible that $v'_i = v_i$ or $v''_i = v_i$. Vertices v'_i and v''_i cannot be adjacent to any v_j with $i \neq j$: that would imply that they are in $N(v_i) \cap N(v_j) \subseteq X$. Thus replacing v_i in I_S with v'_i and v''_i would give a strictly larger independent set, contradicting the maximality of I_S.

Let $\underline{H}$ be the primal graph of H^+. For every $1 \leq i \leq k$, let $C_{i,1}, \ldots, C_{i,c_i}$ be the connected sets given by Lemma 5.13 for the graph $\underline{H}[N_i]$. By the definition of these sets, for every $1 \leq i \leq k$ there is a value $1 \leq d_i \leq c_i$ such that the clique $N_i \cap S$ is fully contained in C_{i,d_i}. Furthermore, the connected set $C_{i,d_i} \setminus (N_i \cap S) = C_{i,d_i} \setminus S$ is contained in a connected component of $\underline{H}[N_i \setminus (N_i \cap S)] = \underline{H}[N_i \setminus S]$, which implies that $C_{i,d_i} \setminus S$ is contained in a connected component of $\underline{H} \setminus S$. Thus either every vertex of $C_{i,d_i} \setminus S$ is reachable from A in $\underline{H} \setminus S$, or none of these vertices are reachable. Let us define $a_i = 1$ in the first case and $a_i = 0$ in the second case (if $C_{i,d_i} \subseteq S$, then define arbitrarily).

We show that if the values d_i, a_i ($1 \leq i \leq k$) corresponding to S are known, then the required separator S' can be found. Thus we have to try all possibilities for these values, which adds a factor of $|V(H)|^{O(w)} \cdot 2^{O(w)}$ to the running time.

Suppose that the values of d_i, a_i are given. Let $Z := X \cup \bigcup_{i=1}^{k} C_{i,d_i}$; note that $S \subseteq Z$. We say that a vertex $u \in C_{i,d_i}$ is a *bad vertex* if

- $a_i = 0$ and there is a path P_a from A to u with $P_a \cap Z = \{u\}$, or
- $a_i = 1$ and there is a path P_b from B to u with $P_b \cap Z = \{u\}$.

(It is possible that P_a or P_b consists of only the vertex u; in particular, if $u \in A \cap B$, then u is always a bad vertex.) Observe that S contains every bad vertex u. Indeed, if $u \notin S$ and there is a path P_a as above, then $S \cap P_a = \emptyset$ (since $S \subseteq Z$), thus u is reachable from A, contradicting $a_i = 0$. On the other hand, if $u \notin S$ and there is a path P_b, then u is reachable from B, but $a_i = 1$ implies that it is also reachable from A, contradicting the fact that S is an (A,B)-separator.

A pair $u \in C_{i,d_i}$ and $v \in C_{j,d_j}$ is a *bad pair* if

- there is a path P from u to v with $P \cap Z = \{u, v\}$ and $a_i \neq a_j$.

In this case, S has to contain at least one of u and v: Otherwise $P \cap S = \emptyset$ would mean that u and v are in the same connected component of $H^+ \setminus S$, implying $a_i = a_j$. Thus every fractional edge cover of S is a solution of the following linear program:

$$\begin{aligned} \min \sum_{e \in E(H^+)} x_e & \\ \sum_{\substack{e \in E(H^+) \\ v \in e}} x_e \geq 1 & \qquad \text{for every bad vertex } v \in Z \\ \sum_{\substack{e \in E(H^+) \\ u \in e}} x_e + \sum_{\substack{e \in E(H^+) \\ v \in e}} x_e \geq 1 & \qquad \text{for every bad pair } u, v \in Z \\ x_e \geq 0 & \qquad \text{for every } e \in E(H^+) \end{aligned}$$

Therefore, the optimum of the linear program is at most w. Let $(x_e)_{e \in E(H^+)}$ be a solution of the linear program with cost at most w. Let Y contain those vertices v for which $\sum_{e \in E(H^+): v \in e} x_e \geq 1/2$; clearly, $\rho^*_{H^+}(Y) \leq 2w$. Thus defining $S' := X \cup Y$ gives a set with $\rho^*_{H^+}(Y) \leq w^3/2 + 2w$. Observe that the linear program ensures that Y (and hence S') contains every bad vertex and at least one vertex from each bad pair.

We claim that S' is an (A, B)-separator in H^+. Suppose that there is a path P from $a \in A$ to $b \in B$ in $H^+ \setminus S'$. This path contains at least one vertex of S (since S is an (A, B)-separator), hence it contains at least one vertex of Z. Let $p_1, \ldots, p_r$ be the vertices of $P \cap Z$, ordered as the path is traversed from a to b. Since these vertices cannot be in $X \subseteq S'$, they are in $\bigcup_{i=1}^{k} C_{i,d_i}$. Suppose first that p_1 is not reachable from A in $H^+ \setminus S$. This means that if N_i is the set that contains p_1, then $a_i = 0$. It follows that p_1 is a bad vertex (because of the subpath of P that connects a with p_1), hence $p_1 \in S'$, a contradiction. Let $1 \leq \ell \leq r$ be the largest value such that p_ℓ is reachable from A in $H^+ \setminus S$ and suppose that p_ℓ is in N_i. If $\ell = r$, then p_ℓ is a bad vertex (because of $a_i = 1$ and the subpath of P connecting p_ℓ and b), again a contradiction. Finally, if $\ell < r$, then let N_j be the set that contains $p_{\ell+1}$. The maximality of ℓ implies $a_i = 1$ and $a_j = 0$. Therefore, p_ℓ, $p_{\ell+1}$ is a bad pair (because of the subpath of P connecting these two vertices), and S' contains at least one of these vertices, a contradiction. Thus S' is an (A, B)-separator in H^+ with $\rho^*_{H^+}(S') \leq w^3/2 + 2w$.

In summary, the algorithm consists of the following steps:

1. Construct the hypergraph H^+ (Lemma 5.12).
2. Guess the independent set I_S.
3. Construct the set X and define the sets N_i.
4. Construct the sets $C_{i,j}$ (Lemma 5.13).
5. Guess the values d_i, a_i.
6. Construct Y using an optimum solution of the linear program.
7. Check if $S' := X \cup Y$ is an (A, B)-separator in H.

As discussed above, if there is an (A, B)-separator S in H with $\rho^*_H(S) \leq w$, then it is possible to choose I_S and the values d_i, a_i such that the separator S' computed by the algorithm is an (A, B)-separator in H with $\rho^*_H(S') \leq w^3 + 4w$. Thus if we try all possible $\|H\|^{O(w)} \cdot \|H\|^{O(w)} \cdot 2^{O(w)}$ guesses, then we will find such a separator S' in this case. On the other hand, if none of the guesses

results in the required separator S', then we can correctly conclude that there is a no (A, B)-separator S in H with $\rho^*_H(S) \leq w$. The running time of each step (except the guesses) is polynomial, thus the total running time is $\|H\|^{O(w)}$. □

In the tree decomposition algorithm of Section 5.2.2, we have to find a balanced separation of a set W: We need a partition (A, B) of W such that (1) $\rho^*_H(A)$, $\rho^*_H(B)$ are not too large and (2) there is an (A, B)-separator S such that $\rho^*_H(S)$ is not too large. As we shall see, it follows Lemma 5.11 proved in Section 5.1.3 that such a balanced separation always exists if H has bounded fractional hypertree width. If we want to find such a separation algorithmically, then the main problem is how to find the partition (A, B) of W: If (A, B) is given, then Lemma 5.15 can be used to find an (A, B)-separator whose fractional edge cover number is bounded. Trying all possible partitions of W is not feasible. Fortunately, for the application in Lemma 5.16, we can assume that $\rho^*_H(W)$ is bounded. Instead of trying all possible partitions of W (the number of such partitions can be exponential in the number of vertices), it turns out that it is sufficient to try all possible partitions of an edge cover F of W (the number of such partitions is exponential only in the size of F).

Lemma 5.16. *Let H be a hypergraph with fractional hypertree width at most w and let $W \subseteq V(H)$ be a subset of vertices with $\rho^*_H(W) \leq k$. It is possible to find in time $\|H\|^{O(w+k)}$ a partition (A, B) of W and an (A, B)-separator S with $\rho^*_H(S) \leq w^3 + 4w$ such that $\rho^*_H(A), \rho^*_H(B) \leq \frac{2}{3}k + w$.*

Proof. Since the fractional edge cover number of W is at most k, the greedy algorithm finds an edge cover $F \subseteq E(H)$ of W with $|F| = O(k \log |V(H)|)$ [228]. Our algorithm tries every partition (F_A, F_B) of F, defines $A := W \cap \bigcup F_A$ and $B := W \setminus A$, and checks whether the algorithm of Lemma 5.15 produces an (A, B)-separator S with $\rho^*_H(S) \leq w^3 + 4w$. We show that if H has fractional hypertree width at most w, then at least one partition (F_A, F_B) results in a partition (A, B) and a separator S satisfying the conditions. Trying every possible partition (F_A, F_B) means trying $2^{O(k \log |V(H)|)} = \|H\|^{O(k)}$ possibilities and the algorithm of Lemma 5.15 needs $\|H\|^{O(w)}$ time. Thus the total running time of the algorithm is $\|H\|^{O(k+w)}$.

By Lemma 5.11, there is a set S_0 with $\rho^*_H(S_0) \leq w$ such that $\rho^*_H(C \cap W) \leq k/2$ for every connected component C of $H \setminus S_0$; let $C_1, \ldots, C_d$ be these connected components. (If $d = 0$, then we are trivially done.) Define $W_i := W \cap C_i$ and suppose that the connected components are ordered such that $\rho^*_H(W_i) \geq \rho^*_H(W_j)$ if $i < j$. Since each edge can intersect at most one W_i, the fractional edge cover number of the union of some W_i's is exactly the sum of the corresponding fractional edge cover numbers. Let ℓ be the largest integer (not greater than d) such that $\rho^*_H(\bigcup_{i=1}^{\ell} W_i) \leq \frac{2}{3}k$. We show that $\rho^*_H(\bigcup_{i=\ell+1}^{d} W_i) \leq \frac{2}{3}k$. Suppose that $\ell < d$, otherwise there is nothing to show. Since $\rho^*_H(W_1) \leq k/2$, we have $\ell \geq 1$. We show that $\rho^*_H(\bigcup_{i=1}^{\ell} W_i) \geq k/3$. This is trivially true if $\rho^*(W_1) \geq k/3$. If $\rho^*(W_1) < k/3$, then we argue as follows. The definition of ℓ implies that $\rho^*_H(\bigcup_{i=1}^{\ell+1} W_i) > \frac{2}{3}k$. Since $\rho^*_H(W_{\ell+1}) \leq \rho^*_H(W_1) \leq k/3$, it follows that $\rho^*_H(\bigcup_{i=1}^{\ell} W_i) \geq k/3$. Since there is no edge that intersects more than one W_i, we have $\rho^*_H(\bigcup_{i=1}^{d} W_i) = \rho^*_H(\bigcup_{i=1}^{\ell} W_i) + \rho^*_H(\bigcup_{\ell+1}^{d} W_i)$. Therefore, $\rho^*_H(\bigcup_{i=1}^{d} W_i) \leq \rho^*_H(W) \leq k$ implies $\rho^*_H(\bigcup_{i=\ell+1}^{d} W_i) \leq \frac{2}{3}k$.

Let F_A be the edges of F fully contained in $S_0 \cup \bigcup_{i=1}^{\ell} C_i$ and let $F_B := F \setminus F_A$; observe that the edges of F_B intersect $\bigcup_{i=\ell+1}^{d} C_i$. Let $A := W \cap \bigcup F_A$ and $B := W \setminus A$ be defined as in the algorithm. Since $A \subseteq S_0 \cup (W \cap \bigcup_{i=1}^{\ell} C_i)$, we have $\rho^*_H(A) \leq \rho^*_H(S_0) + \rho^*_H(\bigcup_{i=1}^{\ell} W_i) \leq w + \frac{2}{3}k$. Similarly, $\rho^*_H(B) \leq w + \frac{2}{3}k$. Observe that S_0 is an (A, B)-separator with $\rho^*_H(S_0) \leq w$, thus the algorithm of Lemma 5.15 produces an (A, B)-separator S with $\rho^*_H(S) \leq w^3 + 4w$. Therefore, when the algorithm considers this particular partition (F_A, F_B), then it finds the required partition (A, B) and separator S. □

5.2.2 Finding approximate tree decompositions

We prove Theorem 5.4 in this section: It is possible to approximate fractional hypertree width in a sense that is suitable for the applications. That is, if a class $\mathcal{H}$ of hypergraphs has bounded fractional hypertree width, then there is a polynomial time algorithm producing a tree decomposition with bounded fractional hypertree width for any hypergraph in $\mathcal{H}$. The algorithm uses the balanced separation algorithm of Lemma 5.16. The following statement immediately implies Theorem 5.4.

Theorem 5.17. *Given a hypergraph H and a rational number $w \geq 1$, it is possible in time $\|H\|^{O(w^3)}$ to either*

- *compute a fractional hypertree decomposition of H with width at most $7w^3 + 31w + 7$, or*
- *correctly conclude that* $\mathrm{fhw}(H) > w$.

Proof. We present an algorithm for a more general problem:

> Given a hypergraph H with $\mathrm{fhw}(H) \leq w$ and a set W with $\rho^*_H(W) \leq 6w^3 + 27w + 6$, find a fractional hypertree decomposition $\mathcal{T}$ of width at most $7w^3 + 31w + 7$ such that some bag B of $\mathcal{T}$ contains the set W.

Note that this algorithm implies the existence of the algorithm required by the theorem: If this algorithm is applied to a hypergraph H and $W = \emptyset$, then either it produces a fractional hypertree decomposition of H with the required width or if the output is something else, then we can correctly conclude that $\mathrm{fhw}(H) > w$. The values $6w^3 + 27w + 6$ and $7w^3 + 31w + 7$ might look somewhat arbitrary, but these are the smallest values ensuring that inequalities (5.1) and (5.2) below are true.

If $\rho^*(H) \leq 7w^3 + 31w + 7$, then we are done, as a tree decomposition consisting of a single bag $B = V(H)$ is sufficient. Thus we can assume that $\rho^*(H) \geq 7w^3 + 31w + 7$. By adding arbitrary vertices to W one by one, we can extend W such that $6w^3 + 27w + 6 \leq \rho^*_H(W) < 6w^3 + 27w + 7$. Let us use the algorithm of Lemma 5.16 to find a partition (A, B) of (the nonempty set) W and an (A, B)-separator S with $\rho^*_H(S) \leq w^3 + 4w$. A connected component of $H \setminus S$ cannot intersect both A and B. Let V_1 be the union of S and all the connected components intersecting A; let V_2 be the union of S and the connected components not intersecting A. Let H_1 (resp., H_2) be the subhypergraph of H induced by V_1 (resp., V_2).

First we verify that H_1 and H_2 are proper subhypergraphs of H; in fact, their fractional edge cover number is strictly less than $\rho^*(H)$. Since $\rho^*_H(W) \leq \rho^*_H(W \cap V_1) + \rho^*_H(W \setminus V_1)$ and $\rho^*_H(W \cap V_1) \leq \rho^*_H(A) + \rho^*_H(S)$, we have

$$\begin{aligned}\rho^*_H(W \setminus V_1) &\geq \rho^*_H(W) - (\rho^*_H(A) + \rho^*_H(S)) \\ &\geq \rho^*_H(W) - \frac{2}{3}\rho^*_H(W) - w - \rho^*_H(S) \geq w^3 + 4w + 2. \quad (5.1)\end{aligned}$$

Consider a fractional edge cover γ of H with weight $\rho^*(H)$. Let γ_S be a fractional edge cover of S with weight $\rho^*_H(S)$. Let us define

$$\gamma'(e) = \begin{cases} \gamma(e) & \text{if } e \cap (W \setminus V_1) = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $\mathrm{weight}(\gamma') \leq \mathrm{weight}(\gamma) - (w^3 + 4w + 2)$, since by (5.1), γ has to assign weight at least $w^3 + 4w + 2$ to the edges intersecting $W \setminus V_1$. Now $\gamma' + \gamma_S$ is an edge cover of V_1 (since

edges intersecting $W \setminus V_1$ cannot intersect $V_1 \setminus S$), thus $\rho^*(H_1) \leq \text{weight}(\gamma') + \text{weight}(\gamma_S) \leq \rho^*(H) - (w^3 + 4w + 2) + \rho^*_H(S) \leq \rho^*(H) - 2$. A similar argument shows $\rho^*(H_2) \leq \rho^*(H) - 2$.

Let $W_1 := A \cup S$ and $W_2 := B \cup S$; we have $\rho^*_H(W_1), \rho^*_H(W_2) \leq \frac{2}{3}\rho^*_H(W) + w + \rho^*_H(S) < 6w^3 + 27w + 6$. Since H_1 and H_2 are strictly smaller than H, we can use the algorithm recursively to obtain a tree decomposition $\mathcal{T}_1$ of H_1 where W_1 is contained in some bag B_1, and a tree decomposition $\mathcal{T}_2$ of H_2 where W_2 is contained in some bag B_2. We connect these two tree decomposition by introducing a new bag $B_0 := W \cup S$ that is connected to B_1 and B_2; note that

$$\rho^*_H(B_0) \leq \rho^*_H(W) + \rho^*_H(S) \leq 7w^3 + 31w + 7. \tag{5.2}$$

It is easy to see that the resulting tree decomposition $\mathcal{T}$ is a proper tree decomposition of H and the bag B_0 fully contains W.

Let us estimate the running time of the algorithm. If $\rho^*(H) \leq 7w^3 + 31w + 7$, then the algorithm constructs only a single bag and does not recurse. We prove by induction that if $\rho^*(H) > 7w^3 + 31w + 7$, then the algorithm constructs a tree decomposition with at most $\rho^*(H) - 2w^3 - 8w - 1$ bags. As the time spent constructing a bag is $\|H\|^{O(w^3)}$, this proves that the running time is $\|H\|^{O(w^3)}$.

First we show that

$$\rho^*_H(V_1) + \rho^*_H(V_2) \leq \rho^*(H) + 2w^3 + 8w. \tag{5.3}$$

To see this, consider a fractional edge cover γ of H with weight $\rho^*(H)$ and let γ_S be a fractional edge cover of S with weight at most $w^3 + 4w$. Let us define

$$\gamma_1(e) = \begin{cases} \gamma(e) & \text{if } e \not\subseteq V_2 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \gamma_2(e) = \begin{cases} \gamma(e) & \text{if } e \not\subseteq V_1 \\ 0 & \text{otherwise.} \end{cases}$$

Since every edge is fully contained in either V_1 or V_2, we have $\text{weight}(\gamma_1) + \text{weight}(\gamma_2) \leq \text{weight}(\gamma)$. Furthermore, $\gamma_1 + \gamma_S$ is a fractional edge cover of V_1, and $\gamma_2 + \gamma_S$ is a fractional edge cover of V_2. Now (5.3) follows from $\text{weight}(\gamma_S) \leq w^3 + 4w$. Subtracting $4w^3 + 16w + 2$ from both sides of (5.3), we get

$$(\rho^*(H_1) - 2w^3 - 8w - 1) + (\rho^*(H_2) - 2w^3 - 8w - 1) \leq (\rho^*(H) - 2w^3 - 8w - 1) - 1 \tag{5.4}$$

Suppose that hypergraph H with $\rho^*(H) > 7w^3 + 31w + 7$ is decomposed into H_1 and H_2. The algorithm constructs a tree decomposition $\mathcal{T}$ that is obtained by joining the tree decompositions $\mathcal{T}_1$ and $\mathcal{T}_2$ with a new bag. Thus $|\mathcal{T}| = |\mathcal{T}_1| + |\mathcal{T}_2| + 1$. We have to consider different cases depending on how $\rho^*(H_1)$, $\rho^*(H_2)$ compare with $7w^3 + 31w + 7$. If $\rho^*(H_1), \rho^*(H_2) > 7w^3 + 31w + 7$, then the induction hypothesis and (5.4) shows $|\mathcal{T}| \leq \rho^*(H) - 2w^3 - 8w - 1$. If $\rho^*(H_1), \rho^*(H_2) \leq 7w^3 + 31w + 7$, then $\mathcal{T}$ consists of only 3 bags. Since $\rho^*(H) - 2w^3 - 8w - 1 \geq 5w^3 + 23w + 6 > 3$, the induction statement holds in this case as well. Suppose now that $\rho^*(H_1) > 7w^3 + 31w + 7$ and $\rho^*(H_2) \leq 7w^3 + 31w + 7$. In this case, $|\mathcal{T}| = |\mathcal{T}_1| + 2$. Now $|\mathcal{T}| \leq \rho^*(H) - 2w^3 - 8w - 1$ follows from the induction hypothesis on H_1 and $\rho^*(H_1) \leq \rho^*(H) - 2$ proved earlier. The case when $\rho^*(H_1) \leq 7w^3 + 31w + 7$ and $\rho^*(H_2) > 7w^3 + 31w + 7$ can be proved similarly. □

CHAPTER 6

Constraint Satisfaction Problems with unbounded arities

There is a long line of research devoted to identifying hypergraph properties that make the evaluation of conjunctive queries tractable (see e.g. [116, 122, 123, 211]). The main contribution of this chapter is giving a complete theoretical answer to this question: in a very precise technical sense, we characterize those hypergraph properties that imply tractability for the evaluation of a query. Efficient evaluation of queries is originally a question of database theory; however, it has been noted that the problem can be treated as a constraint satisfaction problem (CSP) and this connection led to a fruitful interaction between the two communities [118, 166, 211]. Most of the literature relevant to the current chapter use the language of constraint satisfaction. Therefore, we now switch to the language of CSPs (see Sections 1.4–1.5 for the connection between CSPs and database queries).

What are those hypergraph properties that make Boolean Conjunctive Query tractable? In the early 80s, it has been noted that acyclicity is one such property [25, 26, 95, 233]. Later, more general properties of this type were identified in the literature: for example, bounded query width [56], bounded hypertree width [116], and bounded fractional hypertree width [126, 184]. Our goal is to find the most general hypergraph property that guarantees an efficient solution for query evaluation.

Our goal is to characterize the "easy" and "hard" hypergraphs from the viewpoint of constraint satisfaction. However, formally speaking, CSP is polynomial-time solvable for every fixed hypergraph H: since H has a constant number k of vertices, every CSP instance with hypergraph H can be solved by trying all $\|I\|^k$ possible combinations on the k variables. It makes more sense to characterize those *classes* of hypergraphs where CSP is easy. Formally, for a class $\mathcal{H}$ of hypergraphs, let $\mathrm{CSP}(\mathcal{H})$ be the restriction of CSP where the hypergraph of the instance is assumed to be in $\mathcal{H}$. For example, we know that if $\mathcal{H}$ is a class of hypergraphs with bounded treewidth (i.e., there is a constant w such that $\mathrm{tw}(H) \leq w$ for every $H \in \mathcal{H}$), then $\mathrm{CSP}(\mathcal{H})$ is polynomial-time solvable.

For the characterization of the complexity of $\mathrm{CSP}(\mathcal{H})$, we can investigate two notions of tractability. $\mathrm{CSP}(\mathcal{H})$ is *polynomial-time solvable* if there is an algorithm solving every instance of $\mathrm{CSP}(\mathcal{H})$ in time $(\|I\|)^{O(1)}$, where $\|I\|$ is the length of the representation of I in the input. The following notion interprets tractability in a less restrictive way: $\mathrm{CSP}(\mathcal{H})$ is *fixed-parameter tractable (FPT)* if there is an algorithm solving every instance I of $\mathrm{CSP}(\mathcal{H})$ in time $f(H)(\|I\|)^{O(1)}$, where f is an arbitrary computable function of the hypergraph H of the instance. Equivalently, the factor $f(H)$ in the definition can be replaced by a factor $f(k)$ depending only on the number k of vertices of H: as the number of hypergraphs on k vertices (without parallel edges) is bounded by a function of k,

the two definitions result in the same notion. The motivation behind this definition is that if the number of variables is assumed to be much smaller than the the domain size, then we can afford even exponential dependence on the number of variables, as long as the dependence on the size of the instance is polynomial. For a more background on fixed-parameter tractability, the reader is referred to the parameterized complexity literature [74, 87, 102, 197].

We introduce a new hypergraph width measure that we call *submodular width.* Small submodular width means that for every monotone submodular function b on the vertices of the hypergraph H, there is a tree decomposition where $b(B)$ is small for every bag B of the decomposition. (This definition makes sense only if we normalize the considered functions: for this reason, we require that $b(e) \leq 1$ for every edge e of H.) The main result of the chapter is showing that bounded submodular width is the property that precisely characterizes the complexity of CSP($\mathcal{H}$):

Theorem 6.1. *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs. Assuming the Exponential Time Hypothesis,* CSP($\mathcal{H}$) *parameterized by H is fixed-parameter tractable if and only if $\mathcal{H}$ has bounded submodular width.*

Publications. This chapter is based on a single-author publication that appeared in *Journal of the ACM* [188] (an extended abstract appeared in the proceedings of the STOC 2010 conference [186]).

6.1 Introduction

Theorem 6.1 has an algorithmic side (algorithm for bounded submodular width) and a complexity side (hardness result for unbounded submodular width). Unlike previous width measures in the literature, where small value of the measure suggests a way of solving CSP($\mathcal{H}$) it is not at all clear how bounded submodular width is of any help. In particular, it is not obvious what submodular functions have to do with CSP instances. The main idea of our algorithm is that a CSP instance can be "split" into a small number of "uniform" CSP instances; for this purpose, we use a partitioning procedure inspired by a result of Alon et al. [12]. More precisely, splitting means that we partition the set of tuples appearing in the constraint relations in a certain way and each new instance inherits only one class of the partition (thus each new instance has the same set of variables as the original). Uniformity means that for any subset $B \subseteq A$ of variables, every solution for the problem restricted to B has roughly the same number of extensions to A. The property of uniformity allows us to bound the logarithm of the number of solutions on the different subsets by a submodular function. Therefore, bounded submodular width guarantees that each uniform instance has a tree decomposition where only a polynomially bounded number of solutions has to be considered in each bag.

Conceptually, our algorithm goes beyond previous decomposition techniques in two ways. First, the tree decomposition that we use depends not only on the hypergraph, but on the actual constraint relations in the instance (we remark that this idea first appeared in [187] in a different context that does not directly apply to our problem). Second, we are not only decomposing the set of variables, but we also split the constraint relations. This way, we can apply different decompositions to different parts of the solution space.

The proof of the complexity side of Theorem 6.1 follows the same high-level strategy as the proof of Theorem 3.2 in Chapter 3. In a nutshell, the argument is the following: if treewidth is large, then there is subset of vertices which is highly connected in the sense that the set does not have a small balanced separator; such a highly connected set implies that there is uniform concurrent flow (i.e., a compatible set of flows connecting every pair of vertices in the set); the paths in the flows can be used to embed the graph of a 3SAT formula; and finally this embedding can be used to reduce 3SAT

to CSP. These arguments build heavily on well-known characterizations of treewidth and results from combinatorial optimization (such as the $O(\log k)$ integrality gap of sparsest cut). The proof of Theorem 6.1 follows this outline, but now no such well-known tools are available: we are dealing with hypergraphs and submodular functions in a way that was not explored before in the literature. Thus we have to build from scratch all the necessary tools. One of the main difficulties of obtaining Theorem 6.1 is that we have to work in three different domains:

- **CSP instances.** As our goal is to investigate the existence of algorithms solving CSP, the most obvious domain is CSP instances. In light of previous results, we are especially interested in algorithms based on tree decompositions. For such algorithms, what matters is the existence of subsets of vertices such that restricting the instance to any of these subsets gives an instance with "small" number of solutions. In order to solve the instance, we would like to find a tree decomposition where every bag is such a small set.

- **Submodular functions.** Submodular width is defined in terms of submodular functions, thus submodular functions defined on hypergraphs is our second natural domain. We need to understand what large submodular width means, that is, what property of the submodular function and the hypergraph makes it impossible to obtain a tree decomposition where every bag has small value.

- **Flows and embeddings in hypergraphs.** In the hardness proof, our goal is to embed the graph of a 3SAT formula into a hypergraph. Thus we need to define an appropriate notion of embedding and study what guarantees the existence of embeddings with suitable properties. As in Chapter 3, we use the paths appearing in flows to construct embeddings. For our purposes, the right notion of flow is a collection of weighted paths where the total weight of the paths intersecting each hyperedge is at most 1. This notion of flows has not been studied in the literature before, thus we need to obtain basic results on such flows, such as exploring the duality between flows and separators.

A key question is how to find connections between these domains. As mentioned above and detailed in Section 6.4, we have a procedure that reduces a CSP instance into a set of uniform CSP instances, and the number of solutions on the different subsets of variables in a uniform CSP instance can be described by a submodular function. This method allows us to move from the domain of CSP instances to the domain of submodular functions. Section 6.5 is devoted to showing that if submodular width of a hypergraph is large, then there is a certain "highly connected" set in the hypergraph. Highly connected set is defined as a property of the hypergraph (it requires the existence of certain flows) and has no longer anything to do with submodular functions. Thus this connection allows us to move from the domain of submodular functions to the study of hypergraphs. In Section 6.6, we show that a highly connected set in a hypergraph means that graphs can be efficiently embedded into the hypergraph. In particular, the graph of a 3SAT formula can be embedded into the hypergraph, which gives us (as shown in Section 6.7) a reduction from 3SAT to CSP($\mathcal{H}$). This connection allows us to move from the domain of embeddings back to the domain of CSP instances. We remark that Sections 6.4–6.7 are written in a self-contained way: only the first theorem of each section is used outside the section.

Why fixed-parameter tractability? We argue that investigating the fixed-parameter tractability of CSP($\mathcal{H}$) is at least as interesting as investigating polynomial-time solvability. In problems coming from our database-theoretic motivation, the size of the hypergraph (that is, the size of the query) is assumed to be much smaller than the input size (which is usually dominated by the size of the database), hence a constant factor in the running time depending only on the number of

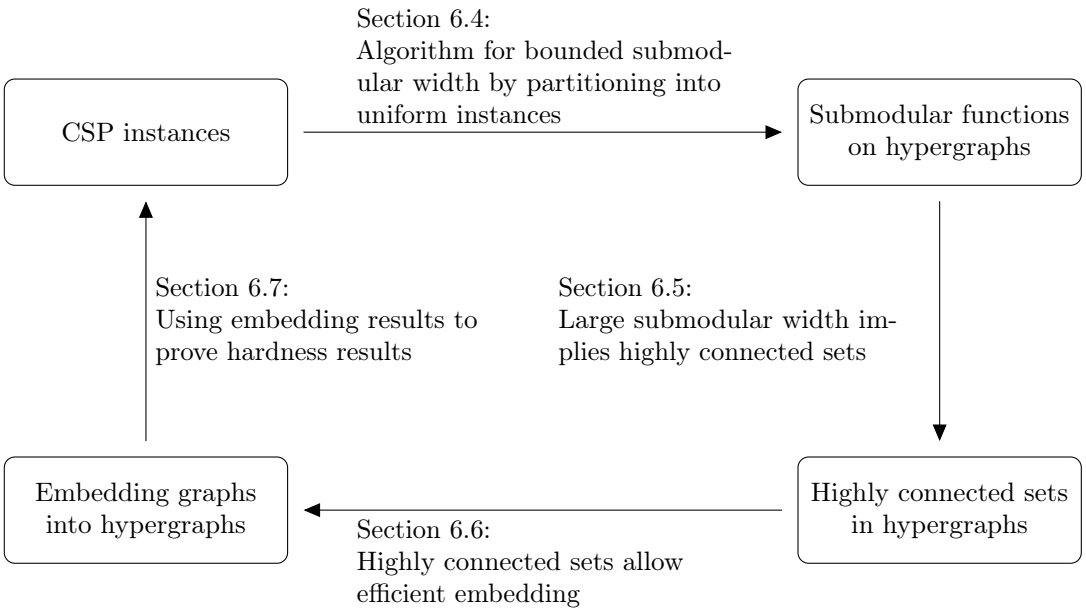

Figure 6.1: Connections between different domains.

variables (or on the hypergraph) is acceptable[1]. Even the STOC 1977 landmark paper of Chandra and Merlin [52], which started the complexity research on conjunctive queries, suggests spending exponential time (in the size of the query) on finding the best possible evaluation order. Furthermore, the notion of fixed-parameter tractability formalizes the usual viewpoint of the literature on conjunctive queries: in the complexity analysis, we should analyze separately the contribution of the query size and the contribution of the database size.

By aiming for fixed-parameter tractability, we can focus more on the core algorithmic question: is there some method for decomposing the space of all solutions in a way that allows efficient evaluation of the query? Some of the progress in this area was made by introducing new decomposition techniques, without showing how to actually find such decompositions. For example, this was the case for the papers introducing query width [56] and fractional hypertree width [126]: it was shown that if a certain type of decomposition is given, then the problem can be solved in polynomial time. In our terminology, these results already show the fixed-parameter tractability of $\#\mathrm{CSP}(\mathcal{H})$ for the classes $\mathcal{H}$ where such decompositions exist (since the time required to find an appropriate decomposition can be bounded by a function of the hypergraph H only), but do not give polynomial-time algorithms. It took some more time and effort to come up with polynomial-time (approximation) algorithms for finding such decompositions [116,184]. While investigating algorithms for finding decompositions give rise to interesting and important problems, they are purely combinatorial problems on graphs and hypergraphs, and no longer has anything to do with query evaluation, constraints, or databases. Thus fixed-parameter tractability gives us a formal way of ignoring these issues and focusing exclusively on the evaluation problem.

On the complexity side, fixed-parameter tractability of $\#\mathrm{CSP}(\mathcal{H})$ seems to be a more robust question than polynomial-time solvability. For example, any polynomial-time reduction to $\#\mathrm{CSP}(\mathcal{H})$ should be able to pick a member of $\mathcal{H}$, thus it seems that polynomial-time reduction to $\#\mathrm{CSP}(\mathcal{H})$ is

[1]This assumption is valid only for evaluation problems (where the problem instance includes a large database) and not for problems that involves only queries, such as the Conjunctive Query Containment problem.

only possible if certain artificial technical conditions are imposed on $\mathcal{H}$ (such as there is an algorithm efficiently generating appropriate members of $\mathcal{H}$). Furthermore, there are classes $\mathcal{H}$ for which CSP($\mathcal{H}$) is polynomial-time equivalent to Log Clique [123], thus we cannot hope to classify CSP($\mathcal{H}$) into polynomial-time solvable and NP-hard cases. Another difficulty in understanding polynomial-time solvability is that it can depend on the "irrelevant" parts of the hypergraph. Suppose for example that there is class $\mathcal{H}$ for which CSP($\mathcal{H}$) is not polynomial-time solvable, but it is fixed-parameter tractable: it can be solved in time $f(H) \cdot (\|I\|)^{O(1)}$. Let $\mathcal{H}'$ be constructed the following way: for every $H \in \mathcal{H}$, class $\mathcal{H}'$ contains a hypergraph H' that is obtained from H by adding a new component that is a path of length $f(H)$. This new path is trivial with respect to the CSP problem, thus any algorithm for CSP($\mathcal{H}$) can be used for CSP($\mathcal{H}'$) as well. Consider an instance I of CSP($\mathcal{H}'$) having hypergraph H', which was obtained from hypergraph H. After taking care of the path, the assumed algorithm for CSP($\mathcal{H}$) can solve this instance in time $f(H) \cdot (\|I\|)^{O(1)}$, which is polynomial in $\|I\|$: instance I contains a representation of H', which has at least $f(H)$ vertices, thus $\|I\|$ is at least $f(H)$. Therefore, CSP($\mathcal{H}'$) is polynomial-time solvable. This example shows that aiming for polynomial-time solvability instead of fixed-parameter tractability might require understanding such subtle, but mostly irrelevant phenomena.

In the hardness results obtained so far, evidence for the non-existence of polynomial-time algorithms is given not in the form of NP-hardness, but by giving evidence that the problem is not even fixed-parameter tractable. For example, in Theorem 3.1, it is a remarkable coincidence that polynomial-time solvability and fixed-parameter tractability are equivalent. However, there is no reason to expect this to remain true in more general cases. Therefore, as discussed above, it makes sense to focus first on understanding the fixed-parameter tractability of the problem.

Organization. For convenience, Section 6.2 collects many of the definitions appearing in this chapter. The reader might want to skim through this at first and refer to appropriate parts of it later. Submodular width and other width measures are defined in Section 6.3. Section 6.4 contains the algorithmic part of the chapter: the algorithm for classes with bounded submodular width. Section 6.5 characterizes large submodular width with highly connected sets, while Section 6.6 uses highly connected sets to find good embeddings in hypergraph. The main hardness result of the chpater is proved in Section 6.7.

6.2 Preliminaries

Let $I = (V, D, C)$ be a CSP instance and let $V' \subseteq V$ be a nonempty subset of variables. If f is a solution of I, then $\mathrm{pr}_{V'} f$ is the *projection* of f to V', which is simply the restriction of the function $f : V \to D$ to $V' \subseteq V$. If R is a set of solutions for I, then we let $\mathrm{pr}_{V'} R = \{\mathrm{pr}_{V'} f \mid f \in R\}$.

The *projection* $\mathrm{pr}_{V'} I$ of I to V' is a CSP $I' = (V', D, C')$, where C' is defined the following way: For each constraint $c = \langle (v_1, \ldots, v_k), R \rangle$ having at least one variable in V', there is a corresponding constraint c' in C'. Suppose that $v_{i_1}, \ldots, v_{i_\ell}$ are the variables among $v_1, \ldots, v_k$ that are in V'. Then the constraint c' is defined as $\langle (v_{i_1}, \ldots, v_{i_\ell}), R' \rangle$, where the relation R' is the projection of R to the coordinates $i_1, \ldots, i_\ell$, that is, R' contains an ℓ-tuple $(d'_1, \ldots, d'_\ell) \in D^\ell$ if and only if there is a k-tuple $(d_1, \ldots, d_k) \in R$ such that $d'_j = d_{i_j}$ for $1 \leq j \leq \ell$. Clearly, if f is a solution of I, then $\mathrm{pr}_{V'} f$ is a solution of $\mathrm{pr}_{V'} I$ (but the converse is not true). For a subset $V' \subseteq V$, we denote by $\mathrm{sol}_I(V')$ the set of all solutions of $\mathrm{pr}_{V'} I$ (which can contain a solution which is not the projection of any solution of I). If the instance I is clear from the context, we drop the subscript.

The *primal graph* (or *Gaifman graph*) of a CSP instance $I = (V, D, C)$ is a graph with vertex set V such that $u, v \in V$ are adjacent if and only if there is a constraint whose scope contains both u and v. The *hypergraph* of a CSP instance $I = (V, D, C)$ is a hypergraph H with vertex set V, where

$e \subseteq V$ is an edge of H if and only if there is a constraint whose scope is e (more precisely, where the scope is an $|e|$-tuple s, whose coordinates form a permutation of the elements of e). For a class $\mathcal{H}$ of graphs, we denote by $\mathrm{CSP}(\mathcal{H})$ the problem restricted to instances whose hypergraph is in $\mathcal{H}$.

Graphs and hypergraphs. If G is a graph or hypergraph, then we denote by $V(G)$ and $E(G)$ the set of vertices and the set of edges of G, respectively. Vertices $u, v \in V(G)$ are *adjacent* if there is an edge $e \in E(G)$ with $u, v \in e$. A set $K \subseteq V(G)$ is a *clique* if the vertices in K are pairwise adjacent. If H is a hypergraph and $V' \subseteq V(H)$, then the *subhypergraph induced by* V' is a hypergraph H' with vertex set S and $\emptyset \subset e' \subseteq V'$ is an edge of H' if and only if there is an edge $e \in E(H)$ with $e \cap V' = e'$. We denote by $H \setminus S$ the subhypergraph of H induced by $V(H) \setminus S$.

Paths, separators, and flows in hypergraphs. A *path* P in hypergraph H is an ordered sequence $v_0, v_1, \ldots, v_r$ of distinct vertices such that v_i and v_{i-1} are adjacent for every $1 \leq i < r$. We distinguish the endpoints of a path: vertex v_0 is the *first endpoint* of P and v_r is the *second endpoint* of P. For a path of length zero, the first and second endpoints coincide. A path is an $X - Y$ *path* if its first endpoint is in X and its second endpoint is in Y. A path $P = v_1 v_2 \ldots v_t$ is *minimal* if there are no shortcuts, i.e., v_i and v_j are not adjacent if $|i - j| > 1$. Note that a minimal path intersects each edge at most twice.

Let H be a hypergraph and $X, Y \subseteq V(H)$ be two (not necessarily disjoint) sets of vertices. An (X, Y)-*separator* is a set $S \subseteq V(H)$ of vertices such that there is no $(X \setminus S) - (Y \setminus S)$ path in $H \setminus S$, or in other words, every $X - Y$ path of H contains at least one vertex of S. In particular, this means that $X \cap Y \subseteq S$.

An assignment $s : E(H) \to \mathbb{R}^+$ is a *fractional* (X, Y)-*separator* if every $X - Y$ path P is *covered* by s, that is, $\sum_{e \in E(H), e \cap P \neq \emptyset} s(e) \geq 1$. The *weight* of the fractional separator s is $\sum_{e \in E(H)} s(e)$.

Let H be a hypergraph and let $\mathcal{P}$ be the set of all paths in H. A *flow* of H is an assignment $f : \mathcal{P} \to \mathbb{R}^+$ such that $\sum_{P \in \mathcal{P}, P \cap e \neq \emptyset} f(P) \leq 1$ for every $e \in E(H)$. The *value* of the flow f is $\sum_{P \in \mathcal{P}} f(P)$. We say that a path P *appears* in flow f, or simply P is a *path of* f if $f(P) > 0$. For some $X, Y \subseteq V(H)$, an (X, Y)-*flow* is a flow f such that only $X - Y$ paths appear in f. A standard LP duality argument shows that the minimum weight of a fractional (X, Y)-separator is equal to the maximum value of an (X, Y)-flow.

If f, f' are flows such that $f'(P) \leq f(P)$ for every path P, then f' is a *subflow* of f. The *sum* of the flows $f_1, \ldots, f_r$ is a mapping that assigns weight $\sum_{i=1}^{r} f_i(P)$ to each path P. Note that the sum of flows is not necessarily a flow itself as the total weight of the paths intersecting a certain edge can be more than 1 in the sum. If the sum of $f_1, \ldots, f_r$ happens to be a flow, then we say that $f_1, \ldots, f_r$ are *compatible.*

Highly connected sets. An important step in understanding various width measures is showing that if the measure is large, then the (hyper)graph contains a highly connected set (in a certain sense). We define here the notion of highly connectedness that will be used in the chapter. First, recall that a *fractional independent set* of a hypergraph H is a mapping $\mu : V(H) \to [0, 1]$ such that $\sum_{v \in e} \mu(v) \leq 1$ for every $e \in E(H)$. We extend functions on the vertices of H to subsets of vertices of H the natural way by setting $\mu(X) := \sum_{v \in X} \mu(v)$, thus we can equivalently say that $\mu : V(H) \to [0, 1]$ is a fractional independent set if and only if $\mu(e) \leq 1$ for every $e \in E(H)$.

Let μ be a fractional independent set of hypergraph H and let $\lambda > 0$ be a constant. We say that a set $W \subseteq V(H)$ is (μ, λ)-*connected* if for any two disjoint sets $A, B \subseteq W$, the minimum weight of a fractional (A, B)-separator is at least $\lambda \cdot \min\{\mu(A), \mu(B)\}$. Note that if W is (μ, λ)-connected, then W is (μ, λ')-connected for every $\lambda' < \lambda$ and every $W' \subseteq W$ is also (μ, λ)-connected. Informally, if W is (μ, λ)-lambda connected for some fractional independent set μ such that $\mu(W)$ is "large", then we call W a highly connected set. For $\lambda > 0$, we denote by $\mathrm{con}_\lambda(H)$ the maximum of $\mu(W)$, taken over every fractional independent set μ and (μ, λ)-connected set W of H. Note that if $\lambda' \leq \lambda$, then $\mathrm{con}_{\lambda'}(H) \geq \mathrm{con}_\lambda(H)$. Throughout this chapter, λ can be thought of as a sufficiently small universal

constant, say, 0.001.

Embeddings. The hardness result presented in this chapter and earlier hardness results for CSP($\mathcal{H}$) [123, 185, 187] are based on embedding some other problem (with a certain graph structure) in a CSP instance whose hypergraph is a member of $\mathcal{H}$. Thus we need appropriate notions of embedding a graph in a (hyper)graph. Let us first recall the definition of minors in graphs. A graph F is a *minor* of G if F can be obtained from G by a sequence of vertex deletions, edge deletions, and edge contractions. The following alternative definition is more relevant from the viewpoint of embeddings: a graph F is a minor of G if there is a mapping ψ that maps each vertex of F to a connected subset of $V(G)$ such that $\psi(u) \cap \psi(v) = \emptyset$ for $u \neq v$, and if $u, v \in V(F)$ are adjacent in F, then there is an edge in $E(G)$ connecting $\psi(u)$ and $\psi(v)$.

A crucial difference between the proof of Theorem 3.1 in [123] and the proof of Theorem 3.2 in Chaper 3 is that the former result is a based on finding a minor embedding of a grid, while the latter result uses a more general notion of embedding where the images of distinct vertices are not necessarily disjoint, but can overlap in a controlled way. We define such embeddings the following way. We say that two sets of vertices $X, Y \subseteq V(H)$ *touch* if either $X \cap Y \neq \emptyset$, or there is an edge $e \in E(H)$ intersecting both X and Y. An *embedding* of graph G into hypergraph H is a mapping ψ that maps each vertex of G to a connected subset of $V(H)$ such that if u and v are adjacent in G, then $\psi(u)$ and $\psi(v)$ touch. The *depth* of a vertex $v \in V(H)$ in embedding ψ is $d_\psi(v) := |\{u \in V(G) \mid v \in \psi(u)\}|$, the number of vertices of G whose images contain v. The *vertex depth* of the embedding is $\max_{v \in V(H)} d_\psi(v)$. Observe that ψ is a minor mapping if and only if it has vertex depth 1. Because in our case we want to control the size of the constraint relations, we need a notion of depth that is sensitive to "what the edges see." We define the depth $d_\psi(e)$ of an edge $e \in E(H)$ as $d_\psi(e) = \sum_{v \in e} d_\psi(e)$ and the *edge depth* to be the maximum of e taken over all edges $e \in E(H)$. Equivalently, we can define the depth of an edge $e \in H$ as $d_\psi(e) = \sum_{v \in V(G)} |\psi(v) \cap e|$, that is, each vertex v contributes $|\psi(v) \cap e|$ to the depth. (A different, perhaps more natural, definition of edge depth would be to define it simply as a maximum number of sets $\psi(v)$ that intersect an edge. Somewhat unexpectedly, most results of this chapter remain true with both notions; see Remarks 6.52–6.53.)

Trivially, for any graph G and hypergraph H, there is an embedding of G into H having vertex depth and edge depth at most $|V(G)|$. If G has m edges and no isolated vertices, then $|V(G)|$ is at most $2m$. We are interested in how much we can gain compared to this trivial solution of depth $O(m)$. We define the *embedding power* $\mathrm{emb}(H)$ to be the maximum (supremum) value of α for which there is an integer m_α such that every graph G with $m \geq m_\alpha$ edges has an embedding into H with edge depth m/α. It might look unmotivated that we define embedding power in terms of the number of edges of G: defining it in terms of the number of vertices might look more natural. However, if we replace the number m of edges with the number n of vertices in the definition, then the worst case occurs if G is a clique on n vertices. Such a definition would describe how well cliques can be embedded, and would give us no information about how sparse graphs can be embedded.

6.3 Width parameters

Treewidth and its various generalizations are defined in this section. We follow the framework of width functions introduced by Adler [7]. A *tree decomposition* of a hypergraph H is a tuple $(T, (B_t)_{t \in V(T)})$, where T is a tree and $(B_t)_{t \in V(T)}$ is a family of subsets of $V(H)$ satisfying the following two conditions: (1) for each $e \in E(H)$ there is a node $t \in V(T)$ such that $e \subseteq B_t$, and (2) for each $v \in V(H)$ the set $\{t \in V(T) \mid v \in B_t\}$ is connected in T. The sets B_t are called the *bags* of the decomposition. Let $f : 2^{V(H)} \to \mathbb{R}^+$ be a function that assigns a nonnegative real number to each nonempty subset

of vertices. The *f-width* of a tree-decomposition $(T, (B_t)_{t\in V(T)})$ is $\max\{f(B_t) \mid t \in V(T)\}$. The *$f$-width* of a hypergraph H is the minimum of the f-widths of all its tree decompositions. In other words, f-width$(H) \leq w$ if and only if there is a tree decomposition of H where $f(B) \leq w$ for every bag B.

The main idea of tree decomposition based algorithms is that if we have a tree decomposition for instance I such that at most C assignments on B_t have to be considered for each bag B_t, then the problem can be solved by dynamic programming in time polynomial in C and $\|I\|$. The various width notions try to guarantee the existence of such decompositions. The simplest such notion, treewidth, can be defined as follows:

Definition 6.2. Let $s(B) = |B| - 1$. The *treewidth* of H is $\mathrm{tw}(H) := s$-width(H).

Further width notions defined in the literature can also be conveniently defined using this setup. A subset $E' \subseteq E(H)$ is an *edge cover* if $\bigcup_{e\in E'} e = V(H)$. The *edge cover number* $\rho(H)$ is the size of the smallest edge cover (here we assume that H has no isolated vertices). For $X \subseteq V(H)$, let $\rho_H(X)$ be the size of the smallest set of edges covering X.

Definition 6.3. The *generalized hypertree width* of H is $\mathrm{ghw}(H) := \rho_H$-width$(H)$.

The original (nongeneralized) definition [116] of hypertree width includes an additional requirement on the decomposition (see Section 5.1), thus it cannot be less than generalized hypertree. However, it is known that hypertree width and generalized hypertree width can differ by at most a constant factor [8].

In Chapter 5, we have seen a further generalization of hypertree width by considering linear relaxations of edge covers. A function $\gamma : E(H) \to [0, 1]$ is a *fractional edge cover* of H if $\sum_{e\in E(H):v\in e} \gamma(e) \geq 1$ for every $v \in V(H)$. The *fractional cover number* $\rho^*(H)$ of H is the minimum of $\sum_{e\in e(H)} \gamma(e)$ taken over all fractional edge covers of H (it is well known that this minimum is achieved by some rational γ). We define $\rho^*_H(X)$ analogously to $\rho_H(X)$: the requirement $\sum_{e:v\in e} \gamma(e) \geq 1$ is restricted to vertices $v \in X$.

Definition 6.4. The *fractional hypertree width* of H is $\mathrm{fhw}(H) := \rho^*_H$-width$(H)$.

A crucial idea appearing in [187] is to make the choice of tree decomposition adaptive: instead of assigning a single decomposition to each hypergraph, we choose the best decomposition based on additional properties of the current instance. Motivated by this idea, we generalize the notion of f-width from a single function f to a class of functions $\mathcal{F}$. Let H be a hypergraph and let $\mathcal{F}$ be an arbitrary (possibly infinite) class of functions that assign nonnegative real numbers to nonempty subsets of vertices of H. The *$\mathcal{F}$-width* of H is $\mathcal{F}$-width$(H) := \sup\{f$-width$(H) \mid f \in \mathcal{F}\}$. Thus if $\mathcal{F}$-width$(H) \leq k$, then for every $f \in \mathcal{F}$, hypergraph H has a tree decomposition with f-width at most k. Note that this tree decomposition can be different for the different functions f. For normalization purposes, we consider only functions f on $V(H)$ that satisfy $f(\emptyset) = 0$ and that are *edge-dominated*, that is, $f(e) \leq 1$ holds for every $e \in E(H)$.

Using these definitions, we can define adaptive width, introduced in [187], as follows. Recall that in Section 6.2, we stated that if μ is a fractional independent set, then μ is extended to subsets of vertices by defining $\mu(X) := \sum_{v\in X} \mu(v)$ for every $X \subseteq V(H)$.

Definition 6.5. The *adaptive width* $\mathrm{adw}(H)$ of a hypergraph H is $\mathcal{F}$-width(H), where $\mathcal{F}$ is the set of all fractional independent sets of H.

A function $f : 2^{V(H)} \to \mathbb{R}$ is *modular* if $f(X) = \sum_{v\in X} c_v$ for some constants c_v ($v \in V(H)$). The function $\mu(X)$ arising from a fractional independent set is clearly a modular and edge dominated

function, in fact, in Definition 6.5 we can equivalently define $\mathcal{F}$ as the set of all nonnegative modular edge-dominated functions on $V(H)$. The main new definition of this chapter is a new width measure, which is obtained by imposing a requirement weaker than modularity on the functions in $\mathcal{F}$ (hence the considered set $\mathcal{F}$ of functions is larger):

Definition 6.6. A function $b : 2^{V(H)} \to \mathbb{R}^+$ is *submodular* if $b(X) + b(Y) \geq b(X \cap Y) + b(X \cup Y)$ holds for every $X, Y \subseteq V(H)$. Given a hypergraph H, let $\mathcal{F}$ contain every edge-dominated monotone submodular function b on $V(H)$ with $b(\emptyset) = 0$. The *submodular width* of hypergraph H is $\mathrm{subw}(H) := \mathcal{F}\text{-width}(H)$.

It is well known that submodular functions can be equivalently characterized by the property that $b(X \cup v) - b(X)$, the *marginal value* of v with respect to X, is a nonincreasing function of X. That is, for every v and $X \subseteq Y$,

$$b(X \cup v) - b(X) \geq b(Y \cup v) - b(Y). \tag{6.1}$$

It is clear that $\mathrm{subw}(H) \geq \mathrm{adw}(H)$: Definition 6.6 considers a larger set of functions than Definition 6.5. Furthermore, we show that $\mathrm{subw}(H)$ is at most the fractional hypertree width $\mathrm{fhw}(H)$. This is a straightforward consequence of the fact that an edge-dominated submodular function is always bounded by the fractional cover number:

Lemma 6.7. *Let H be a hypergraph and b be a monotone edge-dominated submodular function with $b(\emptyset) = 0$. Then $b(S) \leq \rho^*_H(S)$ for every $S \subseteq V(H)$.*

Proof. The statement can be proved along the same lines as the proof of Shearer's Lemma [63] attributed to Radhakrishnan goes. It is sufficient to prove the statement for the case $S = V(H)$: otherwise, we can consider the subhypergraph of H induced by S and the function b restricted to S. Let $\gamma : E(H) \to \mathbb{R}^+$ be a minimum fractional edge cover of S. Let $v_1, \ldots, v_n$ be an arbitrary ordering of $V(H)$ and let $V_i = \{v_1, \ldots, v_i\}$, $V_0 = \emptyset$. For every $e \in E(H)$, we have

$$b(e) = \sum_{v_i \in e} (b(e \cap V_i) - b(e \cap V_{i-1})) \geq \sum_{v_i \in e} (b(V_i) - b(V_{i-1}))$$

(the equality is a simple telescopic sum; the inequality uses (6.1), i.e., the marginal value of v_i with respect to V_{i-1} is not greater than with respect to $e \cap V_{i-1}$).

$$\begin{aligned}\rho^*_H(V(H)) = \sum_{e \in E(H)} \gamma(e) &\geq \sum_{e \in E(H)} \gamma(e) b(e) \geq \sum_{e \in E(H)} \gamma(e) \sum_{v_i \in e} (b(V_i) - b(V_{i-1})) \\ &= \sum_{i=1}^{n} \left((b(V_i) - b(V_{i-1})) \sum_{e \in E(H), v_i \in e} \gamma(e) \right) \geq \sum_{i=1}^{n} (b(V_i) - b(V_{i-1})) = b(V(H))\end{aligned}$$

(in the first inequality, we use that f is edge dominated; in the last inequality, we use that γ is a fractional edge cover). □

Proposition 6.8. *For every hypergraph H,* $\mathrm{subw}(H) \leq \mathrm{fhw}(H)$.

Proof. Let $(T, B_{t \in V(T)})$ be a tree decomposition of H whose ρ^*_H-width is $\mathrm{fhw}(H)$. If b is an edge-bounded monotone submodular function with $b(\emptyset) = 0$, then by Lemma 6.7, $b(B_t) \leq \rho^*_H(B_t) \leq \mathrm{fhw}(H)$ for every bag B_t of the decomposition, i.e., $b\text{-width}(H) \leq \mathrm{fhw}(H)$. This is true for every such function b, hence $\mathrm{subw}(H) \leq \mathrm{fhw}(H)$. □

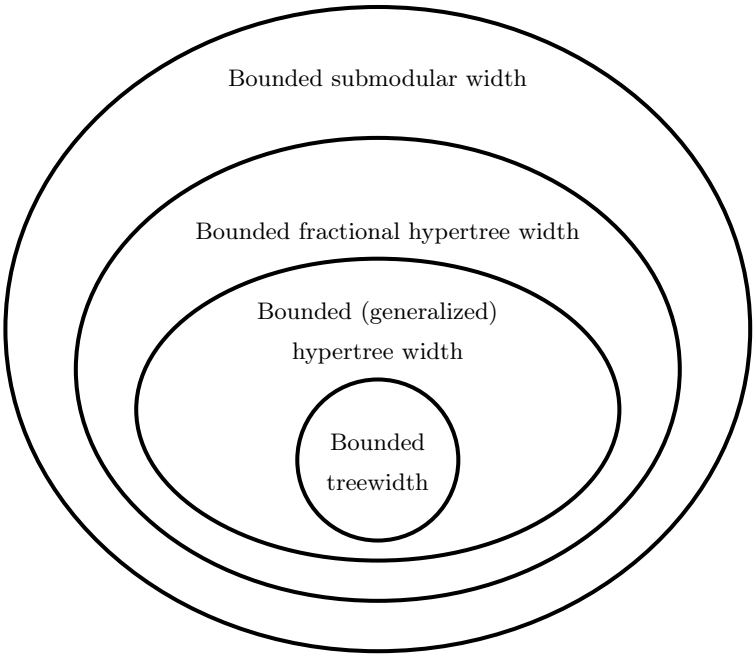

Figure 6.2: Hypergraph properties that make CSP fixed-parameter tractable.

Figure 6.2 shows the relations of the hypergraph properties defined in this section (note that the elements of this Venn diagram are sets of hypergraphs; e.g., the set "bounded treewidth" contains every set $\mathcal{H}$ of hypergraphs with bounded treewidth). As discussed above, all the inclusions in the figure are proper.

Finally, let us remark that there have been investigations of tree decompositions and branch decompositions of submodular functions and matroids in the literature [14, 140–142, 200]. However, in those results the submodular function is a connectivity function: $b(S)$ describes the boundary of S, that is, the cost of separating S from its complement. In our case, $b(S)$ describes the cost of the separator S itself. Therefore, we are in a completely different setting and the previous results cannot be used.

6.4 From CSP instances to submodular functions

In this section, we prove the main algorithmic result of the chapter: $\mathrm{CSP}(\mathcal{H})$ is fixed-parameter tractable if $\mathcal{H}$ has bounded submodular width.

Theorem 6.9. *Let $\mathcal{H}$ be a class of hypergraphs such that* $\mathrm{subw}(H) \leq c_0$ *for every $H \in \mathcal{H}$. Then* $\mathrm{CSP}(\mathcal{H})$ *can be solved in time* $2^{c_0 \cdot 2^{O(|V(H)|)}} \cdot \|I\|^{O(c_0)}$.

The proof of Theorem 6.9 is based on two main ideas:

1. A CSP instance I can be decomposed into a bounded number of "uniform" CSP instances I_1, $\ldots$, I_t (Lemma 6.19). Here uniform means that if $B \subseteq A$ are two sets of variables, then every solution of $\mathrm{pr}_B I_j$ has roughly the same number of extensions to $\mathrm{pr}_A I_j$.

2. If I is a uniform CSP instance, then (the logarithm of) the number of solutions on the different projections of I can be described by an edge-dominated monotone submodular function b

(Lemma 6.20). Therefore, if the hypergraph H of I has bounded submodular width, then it follows that there is a tree decomposition where every bag has a polynomially bounded number of solutions. This means that the existence of a solution can be tested by standard techniques.

While our algorithm is based on these two ideas, the technical implementation is slightly different. First, we can achieve uniformity only on "small sets" of variables. For technical reasons, we have to ensure a certain consistency condition (for example, in order to ensure that the submodular function b is monotone). It follows from the consistency condition that when we find a tree decomposition for a uniform instance such that every bag has a small number of solutions, then this automatically implies that the instance has a solution; we do not even have to use the tree decomposition (see Lemma 6.15).

In Section 6.4.1 we define the notion of consistency that we use and discuss how it can be achieved. Section 6.4.2 describes how the instance can be partitioned into uniform instances. Section 6.4.3 shows how a submodular function can be defined based on a uniform instance, connecting our algorithm to submodular width.

6.4.1 Consistency

Recall from Section 6.2 that $\mathrm{pr}_A I$ is instance I projected to a set A of variables and $\mathrm{sol}_I(A)$ is the set of all solutions of $\mathrm{pr}_A I$. Note that, in general, it is possible that $|\mathrm{sol}_I(S')| > |\mathrm{sol}_I(S)|$ for some $S' \subset S$. In the implementation of the first idea (Lemma 6.19), we guarantee uniformity only to subsets of variables that are "small" in the following hereditary sense

Definition 6.10. Let I be a CSP instance and $M \geq 1$ an integer. We say that $S \subseteq V$ is *M-small* if $|\mathrm{sol}_I(S')| \leq M$ for every $S' \subseteq S$.

It is not difficult to find all the M-small sets, and every solution of the instances projected onto these sets:

Lemma 6.11. *Let $I = (V, D, C)$ be a CSP instance and $M \geq 1$ an integer. There is an algorithm with running time $2^{O(|V|)} \cdot poly(\|I\|, M)$ that finds the set $\mathcal{S}$ of all M-small sets $S \subseteq V$ and constructs $\mathrm{sol}_I(S)$ for each such $S \in \mathcal{S}$.*

Proof. For $i = 1, 2, \ldots, |V|$, we find every M-small set S of size i and construct $\mathrm{sol}_I(S)$. This is trivial to do for $i = 1$. Suppose that we have already found the collection $\mathcal{S}_i$ of all M-small sets of size exactly i. By definition, every size i subset S of an M-small set S of size $i+1$ is an M-small set. Thus we can find every M-small set of size $i+1$ by enumerating every $S \in \mathcal{S}_i$ and checking for every $v \in V \setminus S$ whether $S' := S \cup \{v\}$ is M-small. To check whether S' is M-small, we first check whether every subset of size i is M-small, which is easy to do using the set $\mathcal{S}_i$. Then we construct $\mathrm{sol}_I(S')$: this can be done by enumerating every tuple $s \in \mathrm{sol}_I(S)$ and every extension of s by a new value from D. Thus we need to consider at most $|\mathrm{sol}_I(S)| \cdot |D| \leq M \cdot |D|$ tuples as possible members in $\mathrm{sol}_I(S')$, which means that $\mathrm{sol}_I(S')$ can be constructed in time polynomial in M and $\|I\|$. If $|\mathrm{sol}_I(S')| \leq M$, then we put S' into $\mathcal{S}_{i+1}$. As the collection $\mathcal{S}_i$ contains at most $2^{|V|}$ sets and every operation is polynomial in M and $\|I\|$, the total running time is $2^{O(|V|)} \cdot \mathrm{poly}(\|I\|, M)$. □

We want to avoid dealing with assignments $b \in \mathrm{sol}(B)$ that cannot be extended to any member of $\mathrm{sol}(A)$ for some $A \supseteq B$. Of course, there is no easy way to avoid this in general (or even to detect if there is such a b): for example, if A is the set of all variables, then we would need to check if b can be extended to a solution. Therefore, we require only that there is no such unextendable b if A and B are M-small:

Definition 6.12. A CSP instance I is *M-consistent* if $\mathrm{sol}_I(B) = \mathrm{pr}_B \mathrm{sol}_I(A)$ for all M-small sets $B \subseteq A$.

The notion of M-consistency is very similar to k-consistency, a standard notion in the constraint satisfaction literature [19, 78, 167]. However, we restrict the considered subsets not by the number of variables, but by the number of solutions (more precisely, by considering only M-small sets). Thus the notion of M-consistency could be interpreted in the framework of Greco and Scarcello [121], where consistency is defined with respect to an arbitrary set of views.

Similarly to usual k-consistency, we can achieve M-consistency by throwing away partial solutions that violate the requirements: if we use the algorithm of Lemma 6.11 to find all possible assignments of the M-small sets, then we can check if there is such an unextendable b for some M-small sets A and B. If there is such a b, then we can exclude it from consideration (without losing any solution of the instance) by introducing a new constraint on B. By repeatedly excluding the unextendable assignments, we can avoid all such problems. We say that $I' = (V, D, C')$ is a *refinement* of $I = (V, D, C)$ if for every constraint $\langle s, R\rangle \in C$, there is a constraint $\langle s, R'\rangle \in C'$ such that $R' \subseteq R$.

Lemma 6.13. *Let $I = (V, D, C)$ be a CSP instance and $M \geq 1$ an integer. There is an algorithm with running time $2^{O(|V|)} \cdot poly(\|I\|, M)$ that produces an M-consistent CSP instance I' that is a refinement of I with $\mathrm{sol}(I) = \mathrm{sol}(I')$.*

Proof. Using the algorithm of Lemma 6.11, we can find all the M-small sets and then we can easily check if there are two M-small sets $S \subseteq S'$ violating consistency, i.e., $\mathrm{sol}(S) \not\subseteq \mathrm{pr}_S \mathrm{sol}(S')$. In this case, let s be an $|S|$-tuple whose coordinates contain S in an arbitrary order and let us add the constraint $\langle s, \mathrm{pr}_S \mathrm{sol}(S')\rangle$; it is clear that $\mathrm{sol}(V)$ does not change but $|\mathrm{sol}(S)|$ strictly decreases. We repeat this step until the instance becomes M-consistent. Note that adding the new constraint can make a set M-small that was not M-small before, thus we need to rerun the algorithm of Lemma 6.11. To bound the number of iterations before M-consistency is reached, observe that adding a new constraint does not increase $|\mathrm{sol}(A)|$ for any A and strictly decreases $|\mathrm{sol}(S)|$ for some M-small set S. As there are at most $2^{|V|}$ sets S and $|\mathrm{sol}(S)| \leq M$ for every M-small set S, it follows that this step can be repeated at most $2^{|V|} \cdot M$ times. The size of the instance increases in each step by adding a new constraint with at most M tuples, thus the size of the instance at the end of the process can be still bounded by $2^{O(|V|)} \cdot \mathrm{poly}(\|I\|, M)$. Thus the total time required to ensure that instance I is M-consistent can be bounded by $2^{O(|V|)} \cdot \mathrm{poly}(\|I\|, M)$. □

We want to avoid degenerate cases where there is no solution even for some M-small sets. Consistency implies that it is sufficient to require this for sets of size 1. We say that a CSP instance is *nontrivial* if $\mathrm{sol}(\{v\}) \neq \emptyset$ for every $v \in V$. The following is immediate:

Proposition 6.14. *If I is an M-consistent nontrivial CSP instance, then $\mathrm{sol}(S) \neq \emptyset$ for every M-small set S.*

It is well known that by achieving k-consistency, we can solve CSP instances with treewidth k: the key observation is that if an instance I with treewidth at most k has a k-consistent nontrivial refinement I', then I has a solution. The following lemma adapts this statement to our setting.

Lemma 6.15. *Let $I = (V, D, C)$ be a CSP instance and $M \geq 1$ an integer. Let I' be an M-consistent nontrivial refinement of I. If the hypergraph H of I has a tree decomposition where every bag B is M-small in I', then I has a solution.*

Proof. Suppose that there is such a tree decomposition $(T, (B_t)_{t \in V(T)})$. Assume that T is rooted and for every node $t \in V(T)$, let V_t be the union of the bags that are descendants of t (including B_t).

We claim that every assignment in $\mathrm{sol}_{I'}(B_t)$ can be extended to an assignment of V_t that satisfies every constraint of I whose scope is fully contained in V_t. Applying this statement to the root of T proves that there exists a solution for I. (Recall that every edge of the hypergraph H, and hence the scope of every constraint, is fully contained in one of the bags.)

We prove the claim for every node of T in a bottom up order. The statement is trivial for the leaves. Let $t_1, \ldots, t_\ell$ be the children of t and suppose the claim is true for these nodes. Consider an assignment $g \in \mathrm{sol}_{I'}(B_t)$. Since I' is M-consistent and B_{t_i} is M-small, assignment $g_{|B_t \cap B_{t_i}}$ can be extended to an assignment $g_i \in \mathrm{sol}_{I'}(B_{t_i})$. As the claim is true for node t_i, assignment g_i can be extended to an assignment g'_i of V_{t_i}. The assignments g, g'_1, $\ldots$, g'_ℓ can be combined to obtain an assignment g' on V_t (note that this is well defined: the intersection of V_{t_i} and V_{t_j} is in V_t, which means that a variable appearing in both V_{t_i} and V_{t_j} has the same value in g, g'_i, and g'_j). Furthermore, every edge e of H that is fully contained in V_t is fully contained in at least one of $B_t, V_{t_1}, \ldots, V_{t_\ell}$, and the corresponding assignment among $g, g'_1, \ldots, g'_\ell$ shows that g' satisfies the constraint corresponding to e. □

Note the subtle detail that Lemma 6.15 does not claim that I' has a solution. Furthermore, when Lemma 6.13 creates an M-consistent instance, then it possibly adds many new constraints and the hypergraph of I' can be very dense even if the hypergraph of I has nice structure. However, this is not a problem, as Lemma 6.15 does not require any property on the hypergraph of I'. Note also that every M-small set in I is M-small in I' as well, thus I' has a potentially larger collection of M-small sets, which makes finding the required tree decomposition of the hypergraph H of I easier.

6.4.2 Decomposition into uniform CSP instances

Our algorithm for decomposing a CSP instance into uniform CSP instances is inspired by a combinatorial result of Alon et al. [12], which shows that, for every fixed n, an n-dimensional point set S can be partitioned into $\mathrm{polylog}(|S|)$ classes such that each class is $O(1)$-uniform. We follow the same proof idea: the instance is split into two instances if uniformity is violated somewhere, and we analyze the change of an appropriately defined weight function to bound the number of splits performed. However, the parameter setting is different in our proof: we want to partition into $f(|V|)$ classes, but we are satisfied with somewhat weaker uniformity. Another minor technical difference is that we require uniformity only on the N^c-small sets.

The following definitions gives the precise notion of uniformity that we use:

Definition 6.16. Let $I = (V, D, C)$ be a CSP instance. For $B \subseteq A \subseteq V$ and an assignment $b : B \to D$, let $\mathrm{sol}_I(A|B = b) := \{a \in \mathrm{sol}_I(A) \mid \mathrm{pr}_B\, a = \mathrm{pr}_B\, b\}$, the set of all extensions of b to a solution of $\mathrm{pr}_A\, I$. Let $\max_I(A|B) = \max_{b \in \mathrm{sol}_I(B)} |\mathrm{sol}_I(A|B = b)|$ (if $\mathrm{sol}_I(B) = \emptyset$, then $\max_I(A|B) = 0$). We define $\max_I(A|\emptyset) = |\mathrm{sol}_I(A)|$ and $\max_I(\emptyset|\emptyset) = 1$. We will drop I from the subscript of max if it is clear from the context.

Let us prove two straightforward properties of the function $\max(A|B)$:

Proposition 6.17. *For every $B \subseteq A \subseteq V$ with $\mathrm{sol}(A) \neq \emptyset$ and $C \subseteq V$, we have*

1. $\max(A|B) \geq |\mathrm{sol}(A)|/|\mathrm{sol}(B)|$,
2. $\max(A|B) \geq \max(A \cup C|B \cup C)$.

Proof. If every $b \in \mathrm{sol}(B)$ has at most $\max(A|B)$ extensions to A, then clearly $|\mathrm{sol}(A)|$ is at most $|\mathrm{sol}(B)| \cdot \max(A|B)$, proving the first statement. To show the second statement, consider an $x \in \mathrm{sol}(B \cup C)$ with $\max(A \cup C|B \cup C)$ extensions to $A \cup C$. For any two $y_1, y_2 \in \mathrm{sol}(A \cup C|B \cup C = x)$ with

$y_1 \neq y_2$, we have $\mathrm{pr}_C\, y_1 = \mathrm{pr}_C\, y_2 = \mathrm{pr}_C\, x$, hence y_1 and y_2 can be different only if $\mathrm{pr}_A\, y_1 \neq \mathrm{pr}_A\, y_2$. This means that $\mathrm{pr}_A\, y_1$ and $\mathrm{pr}_A\, y_2$ are two different extensions of $\mathrm{pr}_B\, x$ to A. Therefore,

$$\max(A|B) \geq |\operatorname{sol}(A|B = \mathrm{pr}_B\, x)| \geq |\operatorname{sol}(A \cup C|B \cup C = x)| = \max(A \cup C|B \cup C),$$

what we had to show. □

Notice that (2) in Prop. 6.17 gives a hint that submodularity will be relevant: it is analogous to inequality (6.1) (Section 6.3) expressing that marginal value is larger with respect to a smaller set.

Definition 6.18. We say that $A \subseteq V$ is *c-uniform* (for some integer c) if $\operatorname{sol}(A) \neq \emptyset$ and, for every $B \subseteq A$,

$$\max_I(A|B) \leq c|\operatorname{sol}_I(A)|/|\operatorname{sol}_I(B)|.$$

A CSP instance is (N, c, ϵ)-*uniform* if every N^c-small set is N^ϵ-uniform.

That is, A is c-uniform if every solution of $\operatorname{sol}_I(B)$ has at most c times as many extensions as the average number of extensions. The following lemma states the main combinatorial tool of our algorithm: splitting an instance into a constant number of uniform instances. Note that instances arising from this splitting are more constrained than the original instance, hence it is possible that they contain N^c-small sets that are not N^c-small in the original instance. Therefore, it may happen that the hypergraph of the original instance has no tree decomposition using the N^c-small sets of the original instance, but has a tree decomposition using the N^c-small sets of one of the new instances (and then we can invoke Lemma 6.15 to show that the original instance has a solution).

Lemma 6.19. *Let $I = (V, D, C)$ be a CSP instance, let N an be an integer, and let $c \geq 1$, $\epsilon > 0$ be real numbers. There is an algorithm with running time $2^{2^{O(|V|)} \cdot c/\epsilon} \cdot \mathrm{poly}(\|I\|, N^c)$ that produces a set of (N, c, ϵ)-uniform N^c-consistent nontrivial instances $I_1, \ldots, I_t$ with $0 \leq t \leq 2^{2^{O(|V|)} \cdot c/\epsilon}$, all on the set V of variables, such that*

1. *every solution of I is a solution of exactly one instance I_i,*
2. *for every $1 \leq i \leq t$, instance I_i is a refinement of I.*

Proof. The main step of the algorithm takes a CSP instance I and either makes it (N, c, ϵ)-uniform and N^c-consistent without losing any solutions, or splits it into two instances I_{small}, I_{large}. By applying the main step recursively on I_{small} and I_{large}, we eventually arrive to a set of (N, c, ϵ)-uniform N^c-consistent instances. We will argue that the number of constructed instances is $2^{2^{O(|V|)} \cdot c/\epsilon}$.

In the main step, we first check if the instance is trivial; in this case we can stop with $t = 0$. Otherwise, we invoke the algorithm of Lemma 6.13 to obtain an N^c-consistent refinement of the instance, without losing any solution. Next we check if this N^c-consistent instance I is (N, c, ϵ)-uniform. This can be tested in time $2^{O(|V|)} \cdot \mathrm{poly}(\|I\|, N^c)$ if we use Lemma 6.11 to find all the N^c-small sets and the corresponding sets of solutions. Suppose that N^c-small sets $B \subseteq A$ violate uniformity, that is,

$$\max(A|B) > N^\epsilon|\operatorname{sol}(A)|/|\operatorname{sol}(B)|. \tag{6.2}$$

Let $\operatorname{sol}_{\mathrm{small}}(B)$ contain those tuples b for which $|\operatorname{sol}(A|B = b)| \leq \sqrt{N^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|$ and let $\operatorname{sol}_{\mathrm{large}}(B) = \operatorname{sol}(B) \setminus \operatorname{sol}_{\mathrm{small}}(B)$. Note that $|\operatorname{sol}(A)| \geq |\operatorname{sol}_{\mathrm{large}}(B)| \cdot (\sqrt{N^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|)$ (as every tuple $b \in \operatorname{sol}_{\mathrm{large}}(B)$ has at least $\sqrt{N^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|$ extensions to A), hence

$$|\operatorname{sol}_{\mathrm{large}}(B)| \leq |\operatorname{sol}(B)|/\sqrt{N^\epsilon}. \tag{6.3}$$

Let instance I_{small} (resp., I_{large}) be obtained from I by adding the constraint $\langle B, \text{sol}_{\text{small}}(B)\rangle$ (resp., $\langle B, \text{sol}_{\text{large}}(B)\rangle$). Clearly, the set of solutions of I is the disjoint union of the sets of solutions of I_{small} and I_{large}. This completes the description of the main step.

It is clear that if the recursive procedure stops, then the instances at the leaves of the recursion satisfy the two requirements: the application of Lemma 6.13 does not lose any solution and each resulting instance is N^c-consistent and (N, c, ϵ)-uniform. We show that the height of the recursion tree can be bounded from above by a function $h(|V|, c, \epsilon) = 2^{O(|V|)} \cdot c/\epsilon$ depending only on $|V|$, c, and ϵ; in particular, this shows that the recursive algorithm eventually stops and produces at most $t = 2^{h(|V|,c,\epsilon)} = 2^{2^{O(|V|)} \cdot c/\epsilon}$ instances.

Let us consider a path in the recursion tree starting at the root, and let I^1, I^2, ..., I^p be the corresponding N^c-consistent instances. If a set S is N^c-small in I^j, then it is N^c-small in $I^{j'}$ for every $j' > j$: the main step cannot increase $|\text{sol}(S)|$ for any S. Thus, with the exception of at most $2^{|V|}$ values of j, instances I^j and I^{j+1} have the same N^c-small sets. Let us consider a subpath I^x, ..., I^y such that all these instances have the same N^c-small sets. We show that the length of this subpath is $O(3^{|V|} \cdot c/\epsilon)$, hence $p = O(2^{|V|} \cdot 3^{|V|} \cdot c/\epsilon)$. As this holds for any path starting at the root, we obtain that the height of the recursion tree is $2^{O(|V|)} \cdot c/\epsilon$ and hence $t = 2^{2^{O(|V|)} \cdot c/\epsilon}$.

For the instance I^j, let us define the following weight:

$$W^j = \sum_{\substack{\emptyset \subseteq B \subseteq A \subseteq V \\ A, B \text{ are } N^c\text{-small in } I^j}} \log_2 \max_{I^j}(A|B).$$

We bound the length of the subpath I^x, ..., I^y by analyzing how this weight changes in each step. Observe first that when invoking the algorithm of Lemma 6.13 to find an N^c-consistent refinement, then the weight does not increase: adding new constraints cannot increase $\max(A|B)$ for any $A, B \subseteq V$ and cannot create new N^c-small sets by the assumption on the subpath I^x and I^y. Thus it is sufficient to analyze how the weight decreases in I_{large} and I_{small} compared to I. Note that $0 \leq W^j \leq 3^{|V|} \log_2 N^c = 3^{|V|} \cdot c \log_2 N$: the sum consists of at most $3^{|V|}$ terms and (as A is N^c-small and the instance I^j is N^c-consistent and nontrivial) $\max_{I^j}(A|B)$ is between 1 and N^c. We show that $W^{j+1} \leq W^j - (\epsilon/2) \log_2 N$, which immediately implies that the length of the subpath is $O(3^{|V|} \cdot c/\epsilon)$. Let us inspect how W^{j+1} changes compared to W^j. Since I^j and I^{j+1} have the same N^c-small sets by assumption, no new term can appear in W^{j+1}. It is clear that $\max_{I^{i+1}}(A|B)$ cannot be greater than $\max_{I^i}(A|B)$ for any A, B. Moreover, we show that there is at least one term that strictly decreases. Suppose first that I^{j+1} was obtained from I^j by adding the constraint $\langle B, \text{sol}_{\text{small}}(B)\rangle$. Then

$$\log_2 \max_{I^{j+1}}(A|B) \leq \log_2 \sqrt{N^\epsilon} \frac{|\text{sol}_{I^j}(A)|}{|\text{sol}_{I^j}(B)|} \leq \log_2 \frac{\max_{I^j}(A|B)}{\sqrt{N^\epsilon}} = \log_2 \max_{I^j}(A|B) - (\epsilon/2) \log_2 N,$$

where we have used (6.2) in the second inequality. On the other hand, if I^{j+1} was obtained by adding the constraint $\langle B, \text{sol}_{\text{large}}(B)\rangle$, then

$$\log_2 \max_{I^{j+1}}(B|\emptyset) = \log_2 |\text{sol}_{I^{j+1}}(B)| \leq \log_2(|\text{sol}_{I^j}(B)|/\sqrt{N^\epsilon}) = \log_2 \max_{I^j}(B|\emptyset) - (\epsilon/2) \log_2 N,$$

where the inequality follows from (6.3). In both cases, we get that at least one term decreases by at least $(\epsilon/2) \log_2 N$. □

6.4.3 Uniform CSP instances and submodularity

Assume for a moment that we have a 1-uniform instance I with hypergraph H. Note that by Prop 6.17(1), this means that $\max(A|B) = |\text{sol}(A)|/|\text{sol}(B)|$. Suppose that every constraint

contains at most N tuples and let us define the function $b(S) = \log_N |\operatorname{sol}(S)|$. For every edge $e \in E(H)$, there is a corresponding constraint, which has at most N tuples by the definition of N. Thus $|\operatorname{sol}(e)| \leq N$ and hence $b(e) \leq 1$ for every $e \in E(H)$, that is, b is edge dominated. The crucial observation of this section is that this function b is submodular:

$$\begin{aligned}
b(X) + b(Y) &= \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \frac{|\operatorname{sol}(Y)|}{|\operatorname{sol}(X \cap Y)|} \right) \\
&= \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \cdot \max(Y | X \cap Y) \right) \\
&\geq \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \cdot \max(X \cup Y | X) \right) \\
&= \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \cdot \frac{|\operatorname{sol}(X \cup Y)|}{|\operatorname{sol}(X)|} \right) \\
&= \log_N |\operatorname{sol}(X \cap Y)| + \log_N |\operatorname{sol}(X \cup Y)| \\
&= b(X \cap Y) + b(X \cup Y)
\end{aligned}$$

(the equalities follow from 1-uniformity; the inequality uses Prop. 6.17(2) with $A = Y$, $B = X \cap Y$, $C = X$). Therefore, if the submodular width of H is at most c, then H has a tree decomposition where $b(B) \leq c$ and hence $|\operatorname{sol}(B)| \leq N^c$ for every bag B. Thus we can find a solution of the instance by dynamic programming in time polynomial in N^c.

Lemma 6.19 guarantees some uniformity for the created instances, but not perfect 1-uniformity and only for the N^c-small sets. Thus in Lemma 6.20, we need to define b in a slightly different and more technical way: we add some small terms to correct errors arising from the weaker uniformity and we truncate the function at large values (i.e., for sets that are not N^c-small). Also, we state Lemma 6.20 for an arbitrary hypergraph H, possibly different from the hypergraph of I; then to guarantee that b is edge dominated, we need that $|\operatorname{sol}(e)| \leq N$ for every edge e of H. The reason we need the statement in this form is that in an instance I_i produced by Lemma 6.19 from instance I, the maximum size of a constraint relation in I is an upper bound on $|\operatorname{sol}_{I_i}(e)|$ only if e corresponds to a constraint of I, but we have no such bound if e corresponds to a constraint of I_i not appearing in I.

Lemma 6.20. *Let $I = (V, D, C)$ be a CSP instance and let H be a hypergraph on V (possibly different from the hypergraph of I) such that $|\operatorname{sol}(e)| \leq N$ holds for every $e \in E(H)$. If I is N^c-consistent and (N, c, ϵ^3)-uniform for some $c \geq 1$ and $\epsilon := 1/|V|$, then the following function b is an edge-dominated, monotone, submodular function on H with $b(\emptyset) = 0$:*

$$b(S) := \begin{cases} (1-\epsilon)\log_N |\operatorname{sol}(S)| + 2\epsilon^2|S| - \epsilon^3|S|^2 & \text{if } S \text{ is } N^c\text{-small,} \\ (1-\epsilon)c + 2\epsilon^2|S| - \epsilon^3|S|^2 & \text{otherwise.} \end{cases}$$

Proof. Let $h(S) := 2\epsilon^2|S| - \epsilon^3|S|^2$. The function $h(S)$ is a quadratic function of $|S|$; it is 0 when $|S| = 0$ or $|S| = 2/\epsilon$, hence its maximum is at $|S| = 1/\epsilon = |V(H)|$ with maximum value ϵ. Therefore, $h(S)$ is monotone in the range $0 \leq |S| \leq |V(H)|$. Furthermore, h is a submodular function:

$$\begin{aligned}
&h(X)+h(Y) - h(X \cap Y) - h(X \cup Y) \\
&\quad = 2\epsilon^2(|X| + |Y| - |X \cap Y| - |X \cup Y|) + \epsilon^3(-|X|^2 - |Y|^2 + |X \cap Y|^2 + |X \cup Y|^2) \\
&\quad = \epsilon^3 \left(-(|X \cap Y| + |X \setminus Y|)^2 - (|X \cap Y| + |Y \setminus X|)^2 \right. \\
&\quad + |X \cap Y|^2 + \left. (|X \cap Y| + |X \setminus Y| + |Y \setminus X|)^2 \right) \\
&\quad = 2\epsilon^3 |X \setminus Y| \cdot |Y \setminus X| \geq 0.
\end{aligned}$$

This calculation shows that if $|X \setminus Y|, |Y \setminus X| \geq 1$, then we actually have $h(X) + h(Y) \geq h(X \cap Y) + h(X \cup Y) + 2\epsilon^3$. We will use this extra $2\epsilon^3$ term to dominate the error terms arising from assuming only (N, c, ϵ^3)-uniformity instead of perfect 1-uniformity.

Let us first verify the monotonicity of b. If Y is N^c-small, then every $X \subseteq Y$ is N^c-small, which implies $|\operatorname{sol}(X)| \leq |\operatorname{sol}(Y)|$ as I is N^c-consistent. Therefore, $b(X) \leq b(Y)$ follows from the monotonicity of h. If Y is not N^c small, then $b(Y) = (1-\epsilon)c + h(Y)$ and $b(X) \leq b(Y)$ is clear for every $X \subseteq Y$, no matter whether X is N^c-small or not.

To see that b is edge-dominated, consider an edge $e \in E(H)$. By assumption, $\log_N |\operatorname{sol}(e)| \leq 1$ for every $e \in E(H)$ and hence (using N^c-consistency and $c \geq 1$) e is N^c-small. Thus $b(e) \leq (1-\epsilon) + h(S) \leq 1$, as required.

Finally, let us verify the submodularity of b for some $X, Y \subseteq V$. If $X \subseteq Y$ or $Y \subseteq X$, then there is nothing to show. Thus we can assume that $|X \setminus Y|, |Y \setminus X| \geq 1$. We consider three cases depending on which of X and Y are N^c-small. Suppose first that X and Y are both N^c-small. In this case,

$$\begin{aligned}
b(X) + b(Y) &= (1-\epsilon)\log_N |\operatorname{sol}(X)| + (1-\epsilon)\log_N |\operatorname{sol}(Y)| + h(X) + h(Y) \\
&= (1-\epsilon)\log_N |\operatorname{sol}(X)| + (1-\epsilon)\log_N \left(|\operatorname{sol}(X \cap Y)| \cdot \frac{|\operatorname{sol}(Y)|}{|\operatorname{sol}(X \cap Y)|} \right) \\
&\quad + h(X) + h(Y) \\
&\geq (1-\epsilon)\log_N |\operatorname{sol}(X)| + (1-\epsilon)\log_N |\operatorname{sol}(X \cap Y)| \\
&\quad + (1-\epsilon)\log_N (\max(Y|X \cap Y)/N^{\epsilon^3}) + h(X) + h(Y) \\
&\geq (1-\epsilon)\log_N |\operatorname{sol}(X \cap Y)| + (1-\epsilon)\log_N (|\operatorname{sol}(X)| \max(X \cup Y|X)) \\
&\quad - (1-\epsilon) \cdot \epsilon^3 + h(X \cap Y) + h(X \cup Y) + 2\epsilon^3 \\
&\geq (1-\epsilon)\log_N |\operatorname{sol}(X \cap Y)| + (1-\epsilon)\log_N |\operatorname{sol}(X \cup Y)| + h(X \cap Y) + h(X \cup Y) \\
&\geq b(X \cap Y) + b(X \cup Y)
\end{aligned}$$

(in the first inequality, we used the definition of (N, c, ϵ^3)-uniformity on $X \cap Y$ and Y; in the second inequality, we used the submodularity of h and Prop. 6.17(2) for $A = Y$, $B = X \cap Y$, and $C = X$; in the third inequality, we used Prop. 6.17(1) for $A = X \cup Y$, $B = X$; the last inequality is strict only if $X \cup Y$ is not N^c-small).

For the second case, suppose that, say, X is N^c-small but Y is not. In this case, $X \cap Y$ is N^c-small but $X \cup Y$ is not. Thus

$$\begin{aligned}
b(X) + b(Y) &= (1-\epsilon)\log_N |\operatorname{sol}(X)| + (1-\epsilon)c + h(X) + h(Y) \\
&\geq (1-\epsilon)\log_N |\operatorname{sol}(X \cap Y)| + (1-\epsilon)c + h(X \cap Y) + h(X \cup Y) \\
&= b(X \cap Y) + b(X \cup Y)
\end{aligned}$$

(in the inequality, we used the N^c-consistency on $X \cap Y$ and Y, and the submodularity of h).

Finally, suppose that neither X nor Y is N^c-small. In this case, $X \cup Y$ is not N^c-small either. Now

$$b(X) + b(Y) = 2(1-\epsilon)c + h(X) + h(Y) \geq 2(1-\epsilon)c + h(X \cap Y) + h(X \cup Y) \geq b(X \cap Y) + b(X \cup Y).$$

□

Having constructed the submodular function b as in Lemma 6.20, we can use the argument described at the beginning of the section: if H has submodular width at most $(1-\epsilon)c$, then there is

a tree decomposition where every bag is N^c-small, and we can use this tree decomposition to find a solution. In fact, by Lemma 6.15, in this case N^c-consistency implies that every nontrivial instance has a solution.

Proof (of Theorem 6.9). Let I be an instance of $\mathrm{CSP}(\mathcal{H})$ having hypergraph $H \in \mathcal{H}$. We decide the solvability of I the following way. Let $N \leq \|I\|$ be the size of the largest constraint relation in I, i.e., every constraint has at most N satisfying assignments. Trivially, we have $|\operatorname{sol}_I(e)| \leq N$ for every $e \in E(H)$. Set $\epsilon := 1/|V(H)|$ (we may assume that $|V(H)| \geq 2$), and let $c := c_0/(1-\epsilon) \leq 2c_0$. Let us use the algorithm of Lemma 6.19 to produce the nontrivial N^c-consistent (N, c, ϵ^3)-uniform instances $I_1, \ldots, I_t$. The running time of this step is $2^{2^{O(|V|)} \cdot c/\epsilon} \cdot \mathrm{poly}(\|I\|, N^c)$, which is $2^{c_0 \cdot 2^{O(|V(H)|)}} \cdot \|I\|^{O(c_0)}$.

If $t = 0$, then we can conclude that I has no solution. Otherwise, we argue that I has a solution. Consider any I_i; as I_i is a refinement of I, we have $|\operatorname{sol}_{I_i}(e)| \leq |\operatorname{sol}_I(e)| \leq N$ for any $e \in E(H)$. Let us use Lemma 6.20 with I_i and H (the hypergraph of I, not I_i!) to define the edge-dominated monotone submodular function b. By definition of submodular width, H has a tree decomposition $(T, (B_t)_{t \in V(T)})$ such that $b(B_t) \leq \mathrm{subw}(H) \leq c_0 = (1-\epsilon)c$ for every $t \in V(T)$. Since $b(S) \leq (1-\epsilon)c$ implies $|\operatorname{sol}_{I_i}(S)| \leq N^c$ and b is monotone, this means that B_t is N^c-small in I_i for every $t \in V(T)$. Therefore, the conditions of Lemma 6.15 hold, and I has a solution. □

6.5 From submodular functions to highly connected sets

The aim of this section is to show that if a hypergraph H has large submodular width, then there is a large highly connected set in H. Recall that we say that a set W is (μ, λ)-connected for some fractional independent set μ and $\lambda > 0$, if for every disjoint $A, B \subseteq W$, every fractional (A, B)-separator has weight at least $\lambda \cdot \min\{\mu(A), \mu(B)\}$ (see Section 6.2). Equivalently, we can say that for every disjoint $A, B \subseteq W$, there is an (A, B)-flow of value $\lambda \cdot \min\{\mu(A), \mu(B)\}$. Recall also that $\mathrm{con}_\lambda(H)$ denotes the maximum value of $\mu(W)$ taken over every fractional independent set μ and (μ, λ)-connected set W.

The main result of this section allows us to identify a highly connected set if submodular width is large:

Theorem 6.21. *For every sufficiently small constant $\lambda > 0$, the following holds. Let b be an edge-dominated monotone submodular function of H with $b(\emptyset) = 0$. If the b-width of H is greater than $\frac{3}{2}(w+1)$, then $\mathrm{con}_\lambda(H) \geq w$.*

For the proof of Theorem 6.21, we need to show that if there is no tree decomposition where $b(B)$ is small for every bag B, then a highly connected set exists. There is a standard recursive procedure that either builds a tree decomposition or finds a highly connected set (see e.g., [102, Section 11.2]). Simplifying somewhat, the main idea is that if the graph can be decomposed into smaller graphs by splitting a certain set of vertices into two parts, then a tree decomposition for each part is constructed using the algorithm recursively, and the tree decompositions for the parts are joined in an appropriate way to obtain a tree decomposition for the original graph. On the other hand, if the set of vertices cannot be split, then we can conclude that it is highly connected. This high-level idea has been applied for various notions of tree decompositions [8, 184, 198–200], and it turns out to be useful in our context as well. However, we need to overcome two major difficulties:

1. Highly connected set in our context is defined as not having certain *fractional separators* (i.e., weight assignments). However, if we want to build a tree decomposition in a recursive manner, we need *integer separators* (i.e., subsets of vertices) that decompose the hypergraph into smaller parts.

2. Measuring the sizes of sets with a submodular function b can lead to problems, since the size of the union of two sets can be much smaller than the sum of the sizes of the two sets. We need the property that, roughly speaking, removing a "large" part from a set makes it "much smaller." For example, if A and B are components of $H \setminus S$, and both $b(A)$ and $b(B)$ are large, then we need the property that both of them are much smaller than $b(A \cup B)$. Adler [7, Section 4.2] investigates the relation between some notion of highly connected sets and f-width, but assumes that f is additive: if A and B do not touch, then $f(A \cup B) = f(A) + f(B)$. However, for a submodular function b, there is no reason to assume that additivity holds: for example, it very well may be that $b(A) = b(B) = b(A \cup B)$.

To overcome the first difficulty, we have to understand what fractional separation really means. The first question is whether fractional separation is equivalent to some notion of integral separation, perhaps up to constant factors. The first, naive, question is whether a fractional (X, Y)-separator of weight w implies that there are $O(w)$ edges whose union is an (X, Y)-separator, i.e., there is an (X, Y)-separator S with $\rho_H(S) = O(w)$. There is a simple counterexample showing that this is not true. It is well-known that for every integer $k > 0$, there is a hypergraph H_k such that $\rho^*(H_k) = 2$ and $\rho(H_k) = k$. Let V be the set of vertices of H_k and let H'_k be obtained from H_k by extending it with two independent sets X, Y, each of size k, and connecting every vertex of $X \cup Y$ with every vertex of V. It is clear that there is a fractional (X, Y)-separator of weight 2, but every (X, Y)-separator S has to fully contain at least one of X, Y, or V, implying $\rho_{H'}(S) \geq k$.

A less naive question is whether a fractional (X, Y)-separator with weight w in H implies that there exists an (X, Y)-separator S with $\rho^*_H(S) = O(w)$ (or at most $f(w)$ for some function f). It can be shown that this is not true either: using the hypergraph family presented in [187, Section 5], one can construct counterexamples where the minimum weight of a fractional (X, Y)-separator is a constant, but $\rho^*_H(S)$ has to be arbitrarily large for every (X, Y)-separator S (we omit the details).

We will characterize fractional separation in a very different way. We show that if there is a fractional (A, B)-separator of weight w, then there is an (A, B)-separator S with $b(S) = O(w)$ for *every* edge-dominated monotone submodular function b. Note that this separator S can be different for different functions b, so we are not claiming that there is a single (A, B)-separator S that is small in every b. The converse is also true, thus this gives a novel characterization of fractional separation, tight up to a constant factor. This result is the key idea that allows us to move from the domain of submodular functions to the domain of pure hypergraph properties: if there is no (A, B)-separator such that $b(S)$ is small, then we know that there is no small fractional (A, B)-separator, which is a property of the hypergraph H only and has no longer anything to do with the submodular function b.

To overcome the second difficulty, we introduce a transformation that turns a monotone submodular function b on $V(H)$ into a function b^* that encodes somehow the neighborhood structure of H as well. The new function b^* is no longer monotone and submodular, but it has a number of remarkable properties, for example, b^* remains edge dominated and $b^*(S) \geq b(S)$ for every set $S \subseteq V(H)$, implying that b^*-width is not smaller than b-width. The main idea is to prove Theorem 6.21 for b^*-width instead of b-width (note that this makes the statement stronger). Because of the way b^* encodes the neighborhoods, the second difficulty will disappear: for example, it will be true that $b^*(A \cup B) = b^*(A) + b^*(B)$ if there are no edges between A and B, that is, b^* is additive on disjoint components. Lemma 6.26 formulates (in a somewhat technical way) the exact property of b^* that we will need. Furthermore, luckily it turns out that the result mentioned in the previous paragraph remains true with b replaced by b^*: if there is a fractional (A, B)-separator of weight w, then there is an (A, B)-separator S such that not only $b(S)$, but even $b^*(S)$ is $O(w)$.

6.5.1 The function b^*

We define the function b^* the following way. Let H be a hypergraph and let b be a monotone submodular function defined on $V(H)$. Let $S_{V(H)}$ be the set of all permutations of $V(H)$. For a permutation $\pi \in S_{V(H)}$, let $N_\pi^-(v)$ be the neighbors of v preceding v in the ordering π. For $\pi \in S_{V(H)}$ and $Z \subseteq V(H)$, we define

$$\partial b_{\pi,Z}(v) := b(v \cup (N_\pi^-(v) \cap Z)) - b(N_\pi^-(v) \cap Z).$$

In other words, $\partial b_{\pi,Z}(v)$ is the marginal value of v with respect to the set of its neighbors in Z preceding it. We abbreviate $\partial b_{\pi,V(H)}$ by ∂b_π. As usual, we extend the definition to subsets by letting $\partial b_{\pi,Z}(S) := \sum_{v \in S} \partial b_{\pi,Z}(v)$. Furthermore, we define

$$b_\pi(Z) := \partial b_{\pi,Z}(Z) = \sum_{v \in Z} \partial b_{\pi,Z}(v),$$
$$b^*(Z) := \min_{\pi \in S_{V(H)}} b_\pi(Z).$$

Thus $b_\pi(Z)$ is the sum of the marginal values with respect to a given ordering, while $b^*(Z)$ is the smallest possible sum taken over all possible orderings. Let us prove some simple properties of the function b^*. Properties (1)–(3) and their proofs show why b^* was defined this way: $b^*(Z)$ is never smaller than $b(Z)$, but it is still edge dominated. Properties (4)–(5) are technical statements that we will need later.

Proposition 6.22. *Let H be a hypergraph and let b be a monotone submodular function defined on $V(H)$ with $b(\emptyset) = 0$. For every $\pi \in S_{V(H)}$ and $Z \subseteq V(H)$ we have*

1. $b_\pi(Z) \geq b(Z)$,
2. $b^*(Z) \geq b(Z)$,
3. $b^*(Z) = b_\pi(Z) = b(Z)$ *if Z is a clique,*
4. $\partial b_{\pi,Z_1}(v) \leq \partial b_{\pi,Z_2}(v)$ *if $Z_2 \subseteq Z_1$,*
5. $\partial b_\pi(v) \leq \partial b_{\pi,Z}(v)$,
6. $b^*(X \cup Y) \leq b^*(X) + b^*(Y)$.

Proof. (1) We prove the statement by induction on $|Z|$; for $Z = \emptyset$, the claim is true (as $b(\emptyset) = 0$). Otherwise, let v be the last element of Z according to the ordering π. As v is not preceding any element of Z, for every $u \in Z$ we have $N_\pi^-(u) \cap Z = N_\pi^-(u) \cap (Z \setminus v)$, and hence $\partial b_{\pi,Z}(u) = \partial b_{\pi,Z\setminus v}(u)$.

$$\begin{aligned} b_\pi(Z) &= \sum_{u \in Z \setminus v} \partial b_{\pi,Z}(u) + \partial b_{\pi,Z}(v) = \sum_{u \in Z \setminus v} \partial b_{\pi,Z\setminus v}(u) + \partial b_{\pi,Z}(v) \\ &= b_\pi(Z \setminus v) + \partial b_{\pi,Z}(v) \geq b(Z \setminus v) + b(v \cup (N_\pi^-(v) \cap Z)) - b(N_\pi^-(v) \cap Z) \geq b(Z). \end{aligned}$$

In the first inequality, we used the induction hypothesis and the definition of $\partial b_{\pi,Z}(v)$; in the second inequality, we used the submodularity of b: the marginal value of v with respect to $Z \setminus v$ is not greater than with respect to $N_\pi^-(v) \cap Z \subseteq Z \setminus v$.

(2) Follows immediately from (1) and from the definition of b^*.

(3) As $b_\pi(Z) \geq b^*(Z) \geq b(Z)$ (by property (1) and the definition of $b^*(Z)$), we need to prove $b_\pi(Z) = b(Z)$ only. We prove the statement by induction on $|Z|$. As in (1), let v be the last vertex of Z in π. Note that since Z is a clique, $N_\pi^-(v) \cap Z$ is exactly $Z \setminus v$.

$$b_\pi(Z) = \sum_{u \in Z \setminus v} \partial b_{\pi,Z}(u) + \partial b_{\pi,Z}(v) = \sum_{u \in Z \setminus v} \partial b_{\pi,Z\setminus v}(u) + b(v \cup (N_\pi^-(v) \cap Z)) - b(N_\pi^-(v) \cap Z)$$
$$= b_\pi(Z \setminus v) + b(v \cup (Z \setminus v)) - b(Z \setminus v) = b(Z \setminus v) + b(Z) - b(Z \setminus v) = b(Z).$$

(4) Follows from the submodularity of b: $\partial b_{\pi,Z_1}(v)$ is the marginal value of v with respect to $N_\pi^-(v) \cap Z_1$, while $\partial b_{\pi,Z_2}(v)$ is the marginal value of v with respect to the subset $N_\pi^-(v) \cap Z_2$ of $N_\pi^-(v) \cap Z_1$.

(5) Immediate from (4).

(6) Let π_X and π_Y be the orderings such that $b_{\pi_X}(X) = b^*(X)$ and $b_{\pi_Y} = b^*(Y)$. Let us define ordering π such that it starts with the elements of X, in the order of π_X, followed by the elements of $Y \setminus X$, in the order of π_Y, and completed by an arbitrary ordering of $V(H) \setminus (X \cup Y)$. It is clear that for every $v \in X$, we have $\partial b_{\pi,X\cup Y}(v) = \partial b_{\pi_X,X}(v)$. Furthermore, for every $v \in Y \setminus X$, we have $N_{\pi_Y}^-(v) \cap Y \subseteq N_\pi^-(v) \cap (X \cup Y)$: if u is a neighbor of v in Y that precedes it in π_Y, then u is either in X or in $Y \setminus X$; in both cases u precedes v in π. Thus, similarly to (4), we have $\partial b_{\pi,X\cup Y}(v) \leq \partial b_{\pi_Y,Y}(v)$ for every $v \in Y \setminus X$: $\partial b_{\pi,X\cup Y}(v)$ is the marginal value of v with respect to $N_\pi^-(v) \cap (X \cup Y)$, while $\partial b_{\pi_Y,Y}(v)$ is the marginal value of v with respect to the subset $N_{\pi_Y}^-(v) \cap Y$. Now we have

$$b^*(X \cup Y) \leq b_\pi(X \cup Y) = \sum_{v \in X \cup Y} \partial b_{\pi,X\cup Y}(v) \leq \sum_{v \in X} \partial b_{\pi_X,X}(v) + \sum_{v \in Y \setminus X} \partial b_{\pi_Y,Y}(v)$$
$$\leq \sum_{v \in X} \partial b_{\pi_X,X}(v) + \sum_{v \in Y} \partial b_{\pi_Y,Y}(v) = b^*(X) + b^*(Y). \quad \square$$

Prop. 6.22(3) implies that $\partial b_{w,Z}$ can be used to define a fractional independent set:

Lemma 6.23. *Let H be a hypergraph and let b be an edge-dominated monotone submodular function defined on $V(H)$ with $b(\emptyset) = 0$. Let $W \subseteq V(H)$ and let π be an ordering of W. Let us define $\mu(v) = \partial b_{\pi,W}(v)$ for $v \in W$ and $\mu(v) = 0$ otherwise. Then μ is a fractional independent set of H with $\mu(W) = b_\pi(W)$.*

Proof. Let e be an edge of H and let $Z := e \cap W$. We have

$$\mu(e) = \mu(Z) = \partial b_{\pi,W}(Z) \leq \partial b_{\pi,Z}(Z) = b_\pi(Z) = b(Z) \leq 1,$$

where the first inequality follows from Prop. 6.22(4), the last equality follows from Prop. 6.22(3), and the second inequality follows from the fact that b is edge dominated. Furthermore, we have $\mu(W) = \partial b_{\pi,W}(W) = b_\pi(W)$. $\square$

We close this section by proving the main property of b^* that allows us to avoid the second difficulty described at the beginning of Section 6.5. First, although it is not used directly, let us state that b^* is additive on sets that are independent from each other:

Lemma 6.24. *Let H be a hypergraph, let b be an edge-dominated monotone submodular function defined on $V(H)$ with $b(\emptyset) = 0$, and let $A, B \subseteq V(H)$ be disjoint sets such that there is no edge intersecting both A and B. Then $b^*(A \cup B) = b^*(A) + b^*(B)$.*

Proof. By Prop. 6.22(6), we have to show only $b^*(A \cup B) \geq b^*(A) + b^*(B)$. Let π be an ordering of $V(H)$ such that $b_\pi(A \cup B) = b^*(A \cup B)$; we can assume that π starts with the vertices of $A \cup B$. Since there is no edge that intersects both A and B, and no vertex outside $A \cup B$ precedes a vertex $u \in A \cup B$, we have $N^-_\pi(u) \subseteq A$ for every $u \in A$ and $N^-_\pi(u) \subseteq B$ for every $u \in B$. Thus $\partial b_{\pi,A\cup B}(u) = \partial b_{\pi,A}(u)$ for every $u \in A$ and $\partial b_{\pi,A\cup B}(u) = \partial b_{\pi,B}(u)$ for every $u \in B$. Therefore, $b^*(A \cup B) = b_\pi(A \cup B) = b_\pi(A) + b_\pi(B) \geq b^*(A) + b^*(B)$, what we had to show. □

The actual statement that we use is more complicated than Lemma 6.24: there can be edges between A and B, but we assume that there is a small (A, B)-separator. We want to generalize the following simple statement to our setting:

Proposition 6.25. *Let G be a graph, $W \subseteq V(G)$ a set of vertices, $A, B \subseteq W$ two disjoint subsets, and an (A, B)-separator S. If $|S| < |A|, |B|$, then $|(C \cap W) \cup S| < |W|$ for every component C of $G \setminus S$.*

The proof of Prop. 6.25 is easy to see: every component C of $G \setminus S$ is disjoint from either A or B, thus $|C \cap W|$ is at most $|W| - \min\{|A|, |B|\} < |W| - |S|$, implying that $|(C \cap W) \cup S|$ is less than $|W|$. Statements of this form are useful when constructing tree decompositions in a recursive way. In our setting, we want to measure the size of the sets using the function b^*, not by the number of vertices. More precisely, we measure the size of S and $(C \cap W) \cup S$ using b^*, while the size of W, A, and B are measured using the fractional independent set μ defined by Lemma 6.23. The reason for this will be apparent in the proof of Lemma 6.34: we want to claim that if such a separator S does not exist for any $A, B \subseteq W$, then W is a (μ, λ)-connected set for this fractional independent set μ.

Lemma 6.26. *Let H be a hypergraph, let b be an edge-dominated monotone submodular function defined on $V(H)$ with $b(\emptyset) = 0$ and let W be a set of vertices. Let π_W be an ordering of $V(H)$, and let $\mu(v) := \partial b_{\pi_W,W}(v)$ for $v \in W$ and $\mu(v) = 0$ otherwise. Let $A, B \subseteq W$ be two disjoint sets, and let S be an (A, B)-separator. If $b^*(S) < \mu(A), \mu(B)$, then $b^*((C \cap W) \cup S) < \mu(W)$ for every component C of $H \setminus S$.*

Proof. Let C be a component of $H \setminus S$ and let $Z := (C \cap W) \cup S$. Let π_S be the ordering reaching the minimum in the definition of $b^*(S)$. Let us define the ordering π that starts with S in the order of π_S, followed by $C \cap W$ in the order of π_W, and finished by an arbitrary ordering of the remaining vertices. It is clear that for every $v \in S$, we have $\partial b_{\pi,Z}(v) = \partial b_{\pi_S,S}(v)$. Let us consider a vertex $v \in C \cap W$ and let $u \in W$ be a neighbor of v that precedes it in π_W. Since $v \in C$ and C is a component of $H \setminus S$, either $u \in S$ or $u \in C \cap W$. In both cases, u precedes v in π. This means that $N^-_{\pi_W}(v) \cap W \subseteq N^-_\pi(v) \cap Z$, which implies that $\partial b_{\pi,Z}(v) \leq \partial b_{\pi_W,W}(v) = \mu(v)$ for every $v \in C \cap W$. As S separates A and B, component C intersects at most one of A and B; suppose, without loss of generality, that C is disjoint from A. Thus

$$b^*(Z) \leq b_\pi(Z) = \sum_{v \in S} \partial b_{\pi,Z}(v) + \sum_{v \in C \cap W} \partial b_{\pi,Z}(v) \leq b^*(S) + \mu(C \cap W) < \mu(A) + \mu(W \setminus A) = \mu(W).$$

□

6.5.2 Submodular separation

This section is devoted to understanding what fractional separation means: we show that having a small fractional (A, B)-separator is essentially equivalent to the property that for every edge-dominated submodular function b, there is an (A, B)-separator S such that $b(S)$ is small. The

proof is based on a standard trick that is often used for rounding fractional solutions for separation problems: we define a distance function and show by an averaging argument that cutting at some distance t gives a small separator. However, in our setting, we need significant new ideas to make this trick work: the main difficulty is that the cost function b is defined on *subsets* of vertices and is not a modular function defined by the cost of vertices. To overcome this problem, we use the definitions in Section 6.5.1 (in particular, the function $\partial b_\pi(v)$) to assign a cost to every single vertex.

Theorem 6.27. *Let H be a hypergraph, $X, Y \subseteq V(H)$ two sets of vertices, and $b : V(H) \to \mathbb{R}^+$ an edge-dominated monotone submodular function with $b(\emptyset) = 0$. Suppose that s is a fractional (X,Y)-separator of weight at most w. Then there is an (X,Y)-separator $S \subseteq V(H)$ with $b(S) \leq b^*(S) = O(w)$.*

Proof. The total weight of the edges covering a vertex v is $\sum_{e \in E(H), v \in e} s(e)$; let us define $x(v) := \min\{1, \sum_{e \in E(H), v \in e} s(e)\}$. It is clear that if P is a path from X to Y, then $\sum_{v \in P} x(v) \geq 1$. We define the distance $d(v)$ to be the minimum of $\sum_{v' \in P} x(v')$, taken over all paths from X to v (this means that $d(v) = x(v)$ for every $v \in X$, that is, $d(v) > 0$ is possible for $v \in X$). It is clear that $d(v) \geq 1$ for every $v \in Y$. Let us associate the closed interval $\iota(v) = [d(v) - x(v), d(v)]$ to each vertex v. If v is in X, then the left endpoint of $\iota(v)$ is 0, while if v is in Y, then the right endpoint of $\iota(v)$ is at least 1.

Let u and v be two adjacent vertices in H such that $d(u) \leq d(v)$. It is easy to see that $d(v) \leq d(u) + x(u)$: there is a path P from X to u such that $\sum_{u' \in P} x(u') = d(u)$, thus the path P' obtained by appending v to P has $\sum_{v' \in P'} x(v') = \sum_{u' \in P} x(u') + x(v) = d(u) + x(v)$. Therefore, we have:

Claim 6.28. *If u and v are adjacent, then $\iota(u) \cap \iota(v) \neq \emptyset$.*

The *class* of a vertex $v \in V(H)$ is the largest integer $\kappa(v)$ such that $x(v) \leq 2^{-\kappa(v)}$, and we define $\kappa(v) := \infty$ if $x(v) = 0$. Recall that $x(v) \leq 1$, thus $\kappa(v)$ is nonnegative. The *offset* of a vertex v is the unique value $0 \leq \alpha < 2 \cdot 2^{-\kappa(v)}$ such that $d(v) = i(2 \cdot 2^{-\kappa(v)}) + \alpha$ for some integer i. In other words, if the class is 0, 1, 2, ..., the offset is $d(v)$ modulo 2, 1, 1/2, ..., respectively. Let us define an ordering $\pi = (v_1, \ldots, v_n)$ of $V(H)$ such that

- $\kappa(v)$ is nondecreasing,
- among vertices having the same class, the offset is nondecreasing.

Let directed graph D be the orientation of the primal graph of H such that if v_i and v_j are adjacent and $i < j$, then there is a directed edge $\overrightarrow{v_i v_j}$ in D. Figure 6.3 shows a directed path in D. If P is a directed path in D, then the *width* of P is the length of the interval $\bigcup_{v \in P} \iota(v)$ (note that by Claim 6.28, this union is indeed an interval). The following claim bounds the maximum possible width of a directed path:

Claim 6.29. *If P is a directed path D starting at v, then the width of P is at most $16x(v)$.*

Proof. We first prove that if every vertex of P has the same class $\kappa(v)$, then the width of P is at most $4 \cdot 2^{-\kappa(v)}$. Since the class is nondecreasing along the path, we can partition the path into subpaths such that every vertex in a subpath has the same class and the classes are distinct on the different subpaths. The width of P is at most the sum of the widths of the subpaths, which is at most $\sum_{i \geq \kappa(v)} 4 \cdot 2^{-i} = 8 \cdot 2^{-\kappa(v)} \leq 16x(v)$.

Suppose now that every vertex of P has the same class $\kappa(v)$ as the first vertex v and let $h := 2^{-\kappa(v)}$. As the offset is nondecreasing, path P can be partitioned into two parts: a subpath P_1 containing

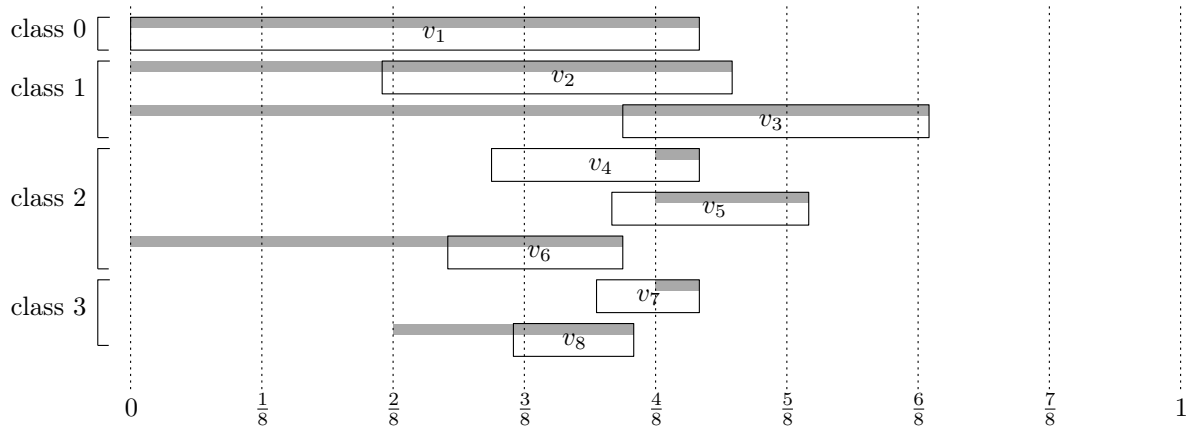

Figure 6.3: The intervals corresponding to a directed path $v_1, \ldots, v_8$. The shaded lines show the offsets of the vertices.

vertices with offset less than h, followed by a subpath P_2 containing vertices with offset at least h (one of P_1 and P_2 can be empty). See Figure 6.4 for examples. We show that each of P_1 and P_2 has width at most $2h$, which implies that the width of P is at most $4h$. Observe that if $u \in P_1$ and $\iota(u)$ contains a point $i \cdot 2h - h$ for some integer i, then, considering $x(u) \leq h$ and the bounds on the offset of u, this is only possible if $\iota(u) = [i \cdot 2h - h, i \cdot 2h]$, i.e., $i \cdot 2h - h$ is the left endpoint of $\iota(u)$. Thus if $I_1 = \bigcup_{u \in P_1} \iota(u)$ contains $i \cdot 2h - h$, then it is the left endpoint of I_1. Therefore, I_1 can contain $i \cdot 2h - h$ for at most one value of i, which immediately implies that the length of I_1 is at most $2h$.

We argue similarly for P_2. If $u \in P_2$, then $\iota(u)$ can contain the point $i \cdot 2h$ only if $\iota(u) = [i \cdot 2h, i \cdot 2h + h]$. Thus if $I_2 = \bigcup_{u \in P_2} \iota(u)$ contains $i \cdot 2h$, then it is the left endpoint of I_2. We get that I_2 can contain $i \cdot 2h$ for at most one value of i, which immediately implies that the width of I_2 is at most $2h$. This concludes the proof of Claim 6.29. ⌟

Let $c(v) := \partial b_\pi(v)$.

Claim 6.30. $\sum_{v \in V(H)} x(v)c(v) \leq w$.

Proof. Let us examine the contribution of an edge $e \in E(H)$ with value $s(e)$ to the sum. For every vertex $v \in e$, edge e increases the value $x(v)$ by at most $s(e)$ (the contribution may be less than $s(e)$, since we defined $x(v)$ to be at most 1). Thus the total contribution of edge e is at most

$$s(e) \cdot \sum_{v \in e} c(v) = s(e) \cdot \sum_{v \in e} \partial b_\pi(v) \leq s(e) \cdot \sum_{v \in e} \partial b_{\pi,e}(v) = s(e) b_\pi(e) = s(e) b(e) \leq s(e),$$

where the first inequality follows Prop. 6.22(5); the last equality follows form Prop. 6.22(3); the last inequality follows from the fact that b is edge dominated. Therefore, $\sum_{v \in V(H)} x(v)c(v) \leq \sum_{e \in E(H)} s(e) \leq w$, proving Claim 6.30. ⌟

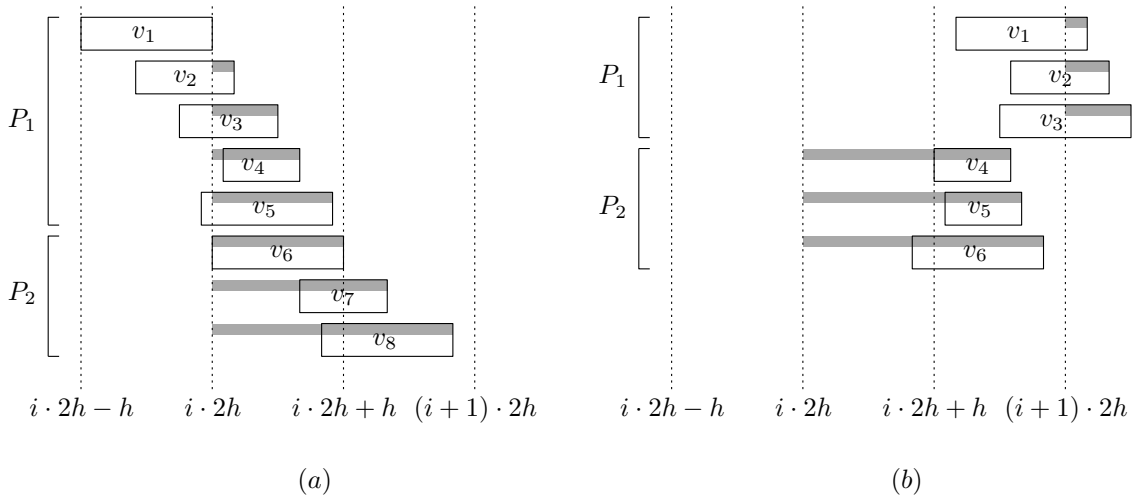

Figure 6.4: Proof of Claim 6.29: Two examples of directed paths where every vertex has the same class κ (and $h := 2^{-\kappa}$). The shaded lines show the offsets of the vertices.

Let S be a set of vertices. We define $\widehat{S}$ to be the "inneighbor closure" of S, that is, the set of all vertices from which a vertex of S is reachable on a directed path in D (in particular, this means that $S \subseteq \widehat{S}$).

Claim 6.31. *For every $S \subseteq V(H)$, $\sum_{v \in \widehat{S}} c(v) = b_\pi(\widehat{S})$.*

Proof. Observe that for any $v \in \widehat{S}$, every inneighbor of v is also in $\widehat{S}$, hence $N^-_\pi(v) \subseteq \widehat{S}$. Therefore, $\partial b_{\pi,\widehat{S}}(v) = \partial b_\pi(v) = c(v)$ and Claim 6.31 follows. ⌟

Let $S(t)$ be the set of all vertices $v \in V(H)$ for which $t \in \iota(v)$. Observe that for every $0 \le t \le 1$, the set $S(t)$ (and hence $\widehat{S}(t)$) separates X from Y. We use an averaging argument to show that there is a $0 \le t \le 1$ for which $b_\pi(\widehat{S}(t))$ is $O(w)$. As $b^*(\widehat{S}(t)) \le b_\pi(\widehat{S}(t))$ by definition, the set $\widehat{S}(t)$ satisfies the requirement of the lemma.

If we are able to show that $\int_0^1 b_\pi(\widehat{S}(t))dt = O(w)$, then the existence of the required t clearly follows. Let $I_v(t) = 1$ if $v \in \widehat{S}(t)$ and let $I_v(t) = 0$ otherwise. If $I_v(t) = 1$, then there is a path P in D from v to a member of $S(t)$. By Claim 6.29, the width of this path is at most $16x(v)$, thus $t \in [d(v) - 16x(v), d(v) + 15x(v)]$. Therefore, $\int_0^1 I_v(t)dt \le 31x(v)$. Now we have

$$\int_0^1 b_\pi(\widehat{S}(t))dt = \int_0^1 \sum_{v \in \widehat{S}(t)} c(v)dt = \int_0^1 \sum_{v \in V(H)} c(v) I_v(t)dt$$
$$= \sum_{v \in V(H)} c(v) \int_0^1 I_v(t)dt \le 31 \sum_{v \in V(H)} x(v)c(v) \le 31w$$

(we used Claim 6.31 in the first equality and Claim 6.30 in the last inequality). □

Although it is not used in this chapter, we can prove the converse of Theorem 6.27 in a very simple way.

Theorem 6.32. *Let H be a hypergraph, and let $X, Y \subseteq V(H)$ be two sets of vertices. Suppose that for every edge-dominated monotone submodular function b on H with $b(\emptyset) = 0$, there is an (X, Y)-separator S with $b(S) \leq w$. Then there is a fractional (X, Y)-separator of weight at most w.*

Proof. If there is no fractional (X, Y)-separator of weight at most w, then by LP duality, there is an (X, Y)-flow F of value greater than w. Let $b(Z)$ be defined as the total weight of the paths in F intersecting Z; it is easy to see that f is a monotone submodular function, and since F is a flow, $b(e) \leq 1$ for every $e \in E(H)$. Thus by assumption, there is an (X, Y)-separator S with $b(S) \leq w$. However, every $X - Y$ path of F intersects (X, Y)-separator S, which implies $b(S) > w$, a contradiction. □

The problem of finding a small separator in the sense of Theorem 6.27 might seem related to submodular function minimization at a first look. We close this section by pointing out that finding an (A, B)-separator S with $b(S)$ small for a given submodular function b is *not* an instance of submodular function minimization, and hence the well-known algorithms (see [149, 150, 213]) cannot be used for this problem. If a submodular function $g(X)$ describes the weight of the *boundary* of X, then finding a small (A, B)-separator is equivalent to minimizing $g(X)$ subject to $A \subseteq X$, $X \cap B = \emptyset$, which can be expressed as an instance of submodular function minimization (and hence solvable in polynomial time). In our case, however, $b(S)$ is the weight of S itself, which means that we have to minimize $g(S)$ subject to S being an (A, B)-separator and this latter constraint cannot be expressed in the framework of submodular function minimization. A possible workaround is to define $\delta(X)$ as the neighborhood of X (the set of vertices outside X adjacent to X) and $b'(X) := b(\delta(S))$; now minimizing $b'(X)$ subject to $A \subseteq X \cup \delta(X)$, $X \cap B = \emptyset$ is the same as finding an (X, Y)-separator S minimizing $b(S)$. However, the function b' is not necessarily a submodular function in general. Therefore, transforming b to b' this way does not lead to a polynomial-time algorithm using submodular function minimization. In fact, it is quite easy to show that finding an (A, B)-separator S with $b(S)$ minimum possible can be an NP-hard problem even if b is a submodular function of very simple form.

Theorem 6.33. *Given a graph G, subsets of vertices X, Y, and collection $\mathcal{S}$ of subsets of vertices, it is NP-hard to find an (X, Y)-separator that intersects the minimum number of members of $\mathcal{S}$.*

Proof. The proof is by reduction from 3-COLORING. Let H be a graph with n vertices and m edges; we identify the vertices of H with the integers from 1 to n. We construct a graph G consisting of $3n+2$ vertices, vertex sets X, Y, and a collection $\mathcal{S}$ of $6m$ sets such that there is an (X, Y)-separator S in G intersecting at most $3m$ members of $\mathcal{S}$ if and only if H is 3-colorable.

The graph G consists of two vertices x, y, and for every $1 \leq i \leq n$, a path $xv_{i,1}v_{i,2}v_{i,3}y$ of length 4 connecting x and y. The collection $\mathcal{S}$ is constructed such that for every edge $ij \in E(H)$ and $1 \leq a, b \leq 3$, $a \neq b$, there is a corresponding set $\{v_{i,a}, v_{j,b}, x, y\}$. Let $X := \{x\}$ and $Y := \{y\}$. Observe that the set $\{v_{i,a}, v_{j,b}\}$ intersects exactly 3 sets of $\mathcal{S}$ if $a \neq b$ and exactly 4 sets of $\mathcal{S}$ if $a = b$.

Let $c : V(H) \to \{1, 2, 3\}$ be a 3-coloring of H. The set $S = \{v_{i,c(i)} \mid 1 \leq i \leq n\}$ is clearly an (X, Y)-separator. For every $ij \in E(H)$, separator S intersects only 3 of the 6 sets $\{v_{i,a}, v_{i,b}, x, y\}$. Therefore, S intersects exactly $3m$ members of $\mathcal{S}$.

Consider now an (X, Y)-separator S intersecting at most $3m$ members of $\mathcal{S}$. Since every member of $\mathcal{S}$ contains both x and y, it follows that $x, y \notin S$. Thus S has to contain at least one internal vertex of every path $xv_{i,1}v_{i,2}v_{i,3}y$. For every $1 \leq i \leq n$, let us fix a vertex $v_{i,c(i)} \in S$. We claim that c is a 3-coloring of H. For every $ij \in E(H)$, S intersects at least 3 of the sets $\{v_{i,a}, v_{i,b}, x, y\}$, and intersects 4 of them if $c(i) = c(j)$. Thus the assumption that S intersects at most $3m$ members of $\mathcal{S}$ immediately implies that c is a proper 3-coloring. □

6.5.3 Obtaining a highly connected set

The following lemma is the same as the main result of Section 6.5 (Theorem 6.21) we are trying to prove, with the exception that b-width is replaced by b^*-width. By Prop 6.22(2), $b^*(S) \geq b(S)$ for every set $S \subseteq V(H)$, thus b^*-width is not less than b-width. Therefore, the following is actually a stronger statement and immediately implies Theorem 6.21.

Lemma 6.34. *For every sufficiently small constant $\lambda > 0$, the following holds. Let b be an edge-dominated monotone submodular function of H with $b(\emptyset) = 0$. If the b^*-width of H is greater than $\frac{3}{2}(w+1)$, then $\mathrm{con}_\lambda(H) \geq w$.*

Proof. Suppose that $\lambda < 1/c$, where c is the universal constant of Lemma 6.27 hidden by the big-O notation. Suppose that $\mathrm{con}_\lambda(H) < w$, that is, there is no fractional independent set μ and (μ, λ)-connected set W with $\mu(W) \geq w$. We show that H has a tree decomposition of b^*-width at most $\frac{3}{2}(w+1)$, or more precisely, we show the following stronger statement:

> For every subhypergraph H' of H and every $W_0 \subseteq V(H')$ with $b^*(W_0) \leq w+1$, there is a tree decomposition of H' having b^*-width at most $\frac{3}{2}(w+1)$ such that W_0 is contained in one of the bags.

We prove this statement by induction on $|V(H')|$. If $b^*(V(H')) \leq \frac{3}{2}(w+1)$, then a decomposition consisting of a single bag proves the statement. Otherwise, let W be a superset of W_0 such that $w \leq b^*(W) \leq w+1$; let us choose a W that is inclusionwise maximal with respect to this property. Observe that there has to be at least one such set: from the fact that $b^*(v) \leq 1$ for every vertex v and from Prop. 6.22(6), we know that adding a vertex increases $b^*(W)$ by at most 1. Since $b^*(V(H')) \geq \frac{3}{2}(w+1)$, by adding vertices to W_0 in an arbitrary order, we eventually find a set W with $b^*(W) \geq w$, and the first such set satisfies $b^*(W) \leq w+1$ as well.

Let π be an ordering of $V(H')$ such that $b_\pi(W) = b^*(W)$. As in Lemma 6.23, let us define the fractional independent set μ by $\mu(v) := \partial b_{\pi,W}(v)$ if $v \in W$ and $\mu(v) = 0$ otherwise. Clearly, we have $\mu(W) = b_\pi(W) = b^*(W) \geq w$.

By assumption, W is not (μ, λ)-connected, hence there are disjoint sets $A, B \subseteq W$ and a fractional (A,B)-separator of weight less than $\lambda \cdot \min\{\mu(A), \mu(B)\}$. Thus by Theorem 6.27, there is an (A,B)-separator $S \subseteq V(H')$ with $b^*(S) < c \cdot \lambda \cdot \min\{\mu(A), \mu(B)\} < \min\{\mu(A), \mu(B)\} \leq \mu(W)/2 \leq (w+1)/2$ (the second inequality follows from the fact that A and B are disjoint subsets of W). Let $C_1, \ldots, C_r$ be the connected components of $H' \setminus S$; by Lemma 6.26, $b^*((C_i \cap W) \cup S) < \mu(W) = b_\pi(W) = b^*(W) \leq w+1$ for every $1 \leq i \leq r$. As $b^*(V(H')) \geq \frac{3}{2}(w+1)$ and $b^*(S) \leq (w+1)/2$, it is not possible that $S = V(H')$, hence $r > 0$. It is not possible that $r = 1$ either: $(C_1 \cap W) \cup S$ is a proper superset of W with b^*-value strictly less than $b^*(W) \leq w+1$, and (as $b^*(V(H')) \geq \frac{3}{2}(w+1)$) we could find a set between $(C_1 \cap W) \cup S$ and $V(H')$ contradicting the maximality of the choice of W. Thus $r \geq 2$, which means that each hypergraph $H'_i := H'[C_i \cup S]$ has strictly fewer vertices than H' for every $1 \leq i \leq r$.

By the induction hypothesis, each H'_i has a tree decomposition $\mathcal{T}_i$ having b^*-width at most $\frac{3}{2}(w+1)$ such that $W_i := (C_i \cap W) \cup S$ is contained in one of the bags. Let B_i be the bag of $\mathcal{T}_i$ containing W_i. We build a tree decomposition $\mathcal{T}$ of H by joining together the tree decompositions $\mathcal{T}_1, \ldots, \mathcal{T}_r$: let $B_0 := W_0 \cup S$ be a new bag that is adjacent to bags $B_1, \ldots, B_r$. It can be easily verified that $\mathcal{T}$ is indeed a tree decomposition of H'. Furthermore, by Prop. 6.22(6), $b^*(B_0) \leq b^*(W_0) + b^*(S) < w + 1 + (w+1)/2 = \frac{3}{2}(w+1)$ and by the assumptions on $\mathcal{T}_1, \ldots, \mathcal{T}_r$, every other bag has b^* value at most $\frac{3}{2}(w+1)$. □

6.6 From highly connected sets to embeddings

The main result of this section is showing that the existence of highly connected sets implies that the hypergraph has large embedding power. Recall from Section 6.2 that W is a (μ, λ)-connected set for some $\lambda > 0$ and fractional independent set μ if for every disjoint $X, Y \subseteq W$, the minimum weight of a fractional (X, Y)-separator is at least $\lambda \cdot \{\mu(X), \mu(Y)\}$. We denote by $\mathrm{con}_\lambda(H)$ the maximum value of $\mu(W)$ taken over every fractional independent set μ and (μ, λ)-connected set W. Recall also that the edge depth of an embedding φ of G into H is the maximum of $\sum_{v \in V(G)} |\varphi(v) \cap e|$, taken over every $e \in E(H)$.

Theorem 6.35. *For every sufficiently small $\lambda > 0$ and hypergraph H, there is a constant $m_{H,\lambda}$ such that every graph G with $m \geq m_{H,\lambda}$ edges has an embedding into H with edge depth $O(m/(\lambda^{\frac{3}{2}} \mathrm{con}_\lambda^{\frac{1}{4}}(H)))$. Furthermore, there is an algorithm that, given G, H, and λ, produces such an embedding in time $f(H, \lambda)n^{O(1)}$.*

In other words, Theorem 6.35 gives a lower bound on the embedding power of H:

Corollary 6.36. *For every sufficiently small $\lambda > 0$ and hypergraph H, $\mathrm{emb}(H) = \Omega(\lambda^{\frac{3}{2}} \mathrm{con}_\lambda^{\frac{1}{4}}(H))$.*

Theorem 6.35 is stated in algorithmic form, since the reduction in the hardness result of Section 6.7 needs to find such embeddings. For the proof, our strategy is similar to the embedding result of Chapter 3: we show that a highly connected set implies that a uniform concurrent flow exists, the paths appearing in the uniform concurrent flow can be used to embed (a blowup of) the line graph of a complete graph, and every graph has an appropriate embedding in the line graph of a complete graph. To make this strategy work, we need generalizations of concurrent flows, multicuts, and multicommodity flows in our hypergraph setting and we need to obtain results that connect these concepts to highly connected sets. Some of these results are similar in spirit to the $O(\sqrt{n})$-approximation algorithms appearing in the combinatorial optimization literature [10,130,134]. However, those approximation algorithms are mostly based on clever rounding of fractional solutions, while in our setting rounding is not an option: as discussed in Section 5, the existence of a fractional (X, Y)-separator of small weight does not imply the existence of a small integer separator. Thus we have to work directly with the fractional solution and use the properties of the highly connected set.

It turns out that the right notion of uniform concurrent flow for our purposes is a collection of flows that connect cliques: that is, a collection $F_{i,j}$ $(1 \leq i < j \leq k)$ of compatible flows, each of value ϵ, such that $F_{i,j}$ is a (K_i, K_j)-flow, where $K_1, \ldots, K_k$ are disjoint cliques. Thus our first goal is to find a highly connected set that can be partitioned into k cliques in an appropriate way.

6.6.1 Highly connected sets with cliques

Let $(X_1, Y_1), \ldots, (X_k, Y_k)$ be pairs of vertex sets such that the minimum weight of a fractional (X_i, Y_i)-separator is s_i. Analogously to multicut problems in combinatorial optimization, we investigate weight assignments that *simultaneously* separate all these pairs. Clearly, the minimum weight of such an assignment is at least the minimum of the s_i's and at most the sum of the s_i's. The following lemma shows that in a highly connected set, such a simultaneous separator cannot be very efficient: roughly speaking, its weight is at least the square root of the sum of the s_i's.

Lemma 6.37. *Let μ be a fractional independent set in hypergraph H and let W be a (μ, λ)-connected set for some $0 < \lambda \leq 1$. Let $(X_1, \ldots, X_k, Y_1, \ldots, Y_k)$ be a partition of W, let $w_i := \min\{\mu(X_i), \mu(Y_i)\} \geq 1/2$, and let $w := \sum_{i=1}^k w_i$. Let $s : E(H) \to \mathbb{R}^+$ be a weight assignment of total weight p such that s is a fractional (X_i, Y_i)-separator for every $1 \leq i \leq k$. Then $p \geq (\lambda/7) \cdot \sqrt{w}$.*

Proof. Let us define the function s' by $s'(e) = 6s(e)$ and let $x(v) := \sum_{e \in E(H), v \in e} s'(e)$. We define the distance $d(u,v)$ to be the minimum of $\sum_{r \in P} x(r)$, taken over all paths P from u to v. It is clear that the triangle inequality holds, i.e., $d(u,v) \leq d(u,z) + d(z,v)$ for every $u, v, z \in V(H)$. If s covers every $u - v$ path, then $d(u,v) \geq 6$: every edge e intersecting a $u - v$ path P contributes at least $s'(e)$ to the sum $\sum_{r \in P} x(r)$ (as e can intersect P in more than one vertices, e can increase the sum by more than $s'(e)$). On the other hand, we claim that if $d(u,v) \geq 2$, then s' covers every $u - v$ path. Clearly, it is sufficient to verify this for minimal paths. Such a path P can intersect an edge e at most twice, hence e contributes at most $2s'(e)$ to the sum $\sum_{r \in P} x(r) \geq 2$, implying that the edges intersecting P have total weight at least 1 in s'.

Suppose for contradiction that $p < (\lambda/7) \cdot \sqrt{w}$, that is, $w > 49p^2/\lambda^2$. As s is an (X_i, Y_i)-separator, we have that $p \geq 1$. Let $A := \emptyset$ and $B := \bigcup_{i=1}^{k} (X_i \cup Y_i)$. Note that $\mu(B) \geq 2 \sum_{i=1}^{k} w_i = 2w$. We will increase A and decrease B while maintaining the invariant condition that the distance of A and B is at least 2 in d. Let T be the smallest integer such that $\sum_{i=1}^{T} w_i > 6p/\lambda$; if there is no such T, then $w \leq 6p/\lambda$, a contradiction. As $w_i \geq 1/2$ for every i, it follows that $T \leq 12p/\lambda + 1 \leq 13p/\lambda$ (since $p \geq 1$ and $\lambda \leq 1$).

For $i = 1, 2, \ldots, T$, we perform the following step. Let X_i' (resp., Y_i') be the set of all vertices of W that are at distance at most 2 from X_i (resp., Y_i). As the distance of X_i and Y_i is at least 6, by the triangle inequality the distance of X_i' and Y_i' is at least 2, hence s' is a fractional (X_i', Y_i')-separator. Since W is (μ, λ)-connected and s' is an assignment of weight $6p$, we have $\min\{\mu(X_i'), \mu(Y_i')\} \leq 6p/\lambda$. If $\mu(X_i') \leq 6p/\lambda$, then let us put X_i (note: *not* X_i') into A and let us remove X_i' from B. The set X_i', which we remove from B, contains all the vertices that are at distance at most 2 from any new vertex in A, hence it remains true that the distance of A and B is at least 2. Similarly, if $\mu(X_i') > 6p/\lambda$ and $\mu(Y_i') \leq 6p/\lambda$, then let us put Y_i into A and let us remove Y_i' from B. Note that we may put a vertex into A even if it was removed from B in an earlier step.

In the i-th step of the procedure, we increase $\mu(A)$ by at least w_i (as $\mu(X_i), \mu(Y_i) \geq w_i$ and these sets are disjoint from the sets already contained in A) and $\mu(B)$ is decreased by at most $6p/\lambda$. Thus at the end of the procedure, we have $\mu(A) \geq \sum_{i=1}^{T} w_i > 6p/\lambda$ and

$$\mu(B) \geq 2w - T \cdot 6p/\lambda > 98p^2/(\lambda^2) - (13p/\lambda)(6p/\lambda) > 6p/\lambda,$$

that is, $\min\{\mu(A), \mu(B)\} > 6p/\lambda$. By the invariant condition, the distance of A and B is at least 2, thus s' is a fractional (A, B)-separator of weight exactly $6p$, contradicting the assumption that W is (μ, λ)-connected. □

In the rest of the section, we need a more constrained notion of flow, where the endpoints "respect" a particular fractional independent set. Let μ_1, μ_2 be fractional independent sets of hypergraph H and let $X, Y \subseteq V(H)$ be two (not necessarily disjoint) sets of vertices. A (μ_1, μ_2)-*demand* (X, Y)-*flow* is an (X, Y)-flow F such that for each $x \in X$, the total weight of the paths in F having first endpoint x is at most $\mu_1(x)$, and similarly, the total weight of the paths in F having second endpoint $y \in Y$ is at most $\mu_2(y)$. Note that there is no bound on the weight of the paths going through an $x \in X$, we only bound the paths whose first/second endpoint is x. The definition is particularly delicate if X and Y are not disjoint, in this case, a vertex $z \in X \cap Y$ can be the first endpoint of some paths and the second endpoint of some other paths, or it can be even both the first and second endpoint of a path of length 0. We use the abbreviation μ-demand for (μ, μ)-demand.

The following lemma shows that if a flow connects a set U with a highly connected set W, then U is highly connected as well ("W can be moved to U"). This observation will be used in the proof of Lemma 6.39, where we locate cliques and show that their union is highly connected, since there is a flow that connects the cliques to a highly connected set.

Lemma 6.38. *Let H be a hypergraph, μ_1, μ_2 fractional independent sets, and $W \subseteq V(H)$ a (μ_1, λ)-connected set for some $0 < \lambda \leq 1$. Suppose that $U \subseteq V(H)$ is a set of vertices and F is a (μ_1, μ_2)-demand (W, U)-flow of value $\mu_2(U)$. Then U is $(\mu_2, \lambda/6)$-connected.*

Proof. Suppose that there are disjoint sets $A, B \subseteq U$ and a fractional (A, B)-separator s of weight $w < (\lambda/6) \cdot \min\{\mu_2(A), \mu_2(B)\}$. (Note that this means $\mu_2(A), \mu_2(B) > 6w/\lambda \geq 6w$.) For a path P, let $s(P) = \sum_{e \in E(H), e \cap P \neq \emptyset} s(e)$ be the total weight of the edges intersecting P. Let $A' \subseteq W$ (resp., $B' \subseteq W$) contain a vertex $v \in W$ if there is a path P in F with first endpoint v and second endpoint in A (resp., B) and $s(P) \leq 1/3$. If $A' \cap B' \neq \emptyset$, then it is clear that there is a path P with $s(P) \leq 2/3$ connecting a vertex of A and a vertex of B via a vertex of $A' \cap B'$, a contradiction. Thus we can assume that A' and B' are disjoint.

Since F is a flow and s has weight w, the total weight of the paths in F with $s(P) \geq 1/3$ is at most $3w$. As the value of F is exactly $\mu_2(U)$, the total weight of the paths in F with second endpoint in A is exactly $\mu_2(A)$. If $s(P) \leq 1/3$ for such a path, then its first endpoint is in A' by definition. Therefore, the total weight of the paths in F with first endpoint in A' is at least $\mu_2(A) - 3w$, which means that $\mu_1(A') \geq \mu_2(A) - 3w \geq \mu_2(A)/2$. Similarly, we have $\mu_1(B') \geq \mu_2(B)/2$. Since W is (μ_1, λ)-connected and s is an assignment with weight less than $(\lambda/6) \cdot \min\{\mu_2(A), \mu_2(B)\} \leq (\lambda/3) \cdot \min\{\mu_1(A'), \mu_1(B')\}$, there is an $A' - B'$ path P with $s(P) < 1/3$. Now the concatenation of an $A' - A$ path P_A having $s(P_A) \leq 1/3$, the path P, and a $B' - B$ path P_B having $s(P_B) \leq 1/3$ forms an $A - B$ path that is not covered by s, a contradiction. □

A μ-demand *multicommodity flow* between pairs $(A_1, B_1), \ldots, (A_r, B_r)$ is a set $F_1, \ldots, F_r$ of compatible flows such that F_i is a μ-demand (A_i, B_i)-flow (recall that a set of flows is compatible if their sum is also a flow, that is, does not violate the edge constraints). The *value* of a multicommodity flow is the sum of the values of the r flows. Let $A = \bigcup_{i=1}^{r} A_i$, $B = \bigcup_{i=1}^{r} B_i$, and let us restrict our attention to the case when $(A_1, \ldots, A_r, B_1, \ldots, B_r)$ is a partition of $A \cup B$. In this case, the maximum value of a μ-demand multicommodity flow between pairs $(A_1, B_1), \ldots, (A_r, B_r)$ can be expressed as the optimum values of the primal and dual linear programs in Figure 6.5.

The following lemma shows that if $\mathrm{con}_\lambda(H)$ is sufficiently large, then there is a highly connected set that has the additional property that it is the union of k cliques $K_1, \ldots, K_k$ with $\mu(K_i) \geq 1/2$ for every clique. The high-level idea of the proof is the following. Take a (μ, λ)-connected set W with $\mu(W) = \mathrm{con}_\lambda(H)$ and find a large multicommodity flow between some pairs $(A_1, B_1), \ldots, (A_r, B_r)$ in W. Consider the dual solution y. By complementary slackness, every edge with nonzero value in y covers exactly 1 unit of the multicommodity flow. If most of the weight of the dual solution is on the edge variables, then we can choose k edges that cover at least $\Omega(k)$ units of flow. These edges are connected to W by a flow, and therefore by Lemma 6.38 the union of these edges is also highly connected and obviously can be partitioned into a small number cliques.

There are two things that can go wrong with this argument. First, it can happen that the dual solution assigns most of the weight to the vertex variables $y(u)$, $y(v)$ ($u \in A$, $v \in B$). The cost of covering the $A_i - B_i$ paths using vertex variables only is $\min\{\mu(A_i), \mu(B_i)\}$, thus this case is only possible if the value of the dual (and hence the primal) solution is close to $\sum_{i=1}^{r}(\min\{\mu(A_i), \mu(B_i)\})$. To avoid this situation, we want to select the pairs (A_i, B_i) such that they are only "moderately connected": there is a fractional (A_i, B_i)-separator of weight $2\lambda \min\{\mu(A_i), \mu(B_i)\}$, that is, at most twice the minimum possible. This means that the weight of the dual solution is at most $2\lambda \sum_{i=1}^{r}(\min\{\mu(A_i), \mu(B_i)\})$, which is much less than $\sum_{i=1}^{r}(\min\{\mu(A_i), \mu(B_i)\}$ (if λ is small). If we are not able to find sufficiently many such pairs, then we argue that a larger highly connected set can be obtained by scaling μ by a factor of 2. More precisely, we show that there is a large subset $W' \subseteq W$ that is $(2\mu, \lambda)$-connected and $2\mu(W') > \mathrm{con}_\lambda(H)$, a contradiction (a technical difficulty here that we have to make sure first that 2μ is also a fractional independent set).

Primal LP

$$\text{maximize} \quad \sum_{i=1}^{r} \sum_{\substack{u\in A_i, v\in B_i \\ P\in\mathcal{P}_{uv}}} x(P)$$

$$\text{s.t.}$$

$$\sum_{i=1}^{r} \sum_{\substack{u\in A_i, v\in B_i \\ P\in\mathcal{P}_{uv}, P\cap e\neq\emptyset}} x(P) \le 1 \quad \forall e \in E(H)$$

$$\sum_{v\in B_i, P\in\mathcal{P}_{uv}} x(P) \le \mu(u) \quad \forall 1 \le i \le r, u \in A_i$$

$$\sum_{u\in A_i, P\in\mathcal{P}_{uv}} x(P) \le \mu(v) \quad \forall 1 \le i \le r, v \in B_i$$

$$x(P) \ge 0 \quad \begin{array}{l}\forall 1 \le i \le r, \\ u \in A_i, v \in B_i, \\ P \in \mathcal{P}_{uv}\end{array}$$

Dual LP

$$\text{minimize} \quad \sum_{e\in e(H)} y(e) + \sum_{u\in A} \mu(u)y(u) + \sum_{v\in B} \mu(v)y(v)$$

$$\text{s.t.}$$

$$\sum_{\substack{e\in E(H), \\ e\cap P\neq\emptyset}} y(e) + y(u) + y(v) \ge 1 \quad \begin{array}{l}\forall 1 \le i \le r, \\ u \in A_i, v \in B_i, \\ P \in \mathcal{P}_{uv}\end{array}$$

$$y(e) \ge 0 \quad \forall e \in E(H)$$

$$y(u) \ge 0 \quad \forall u \in A$$

$$y(v) \ge 0 \quad \forall v \in B$$

Figure 6.5: Primal and dual linear programs for μ-demand multicommodity flow between pairs $(A_1, B_1), \ldots, (A_r, B_r)$. We denote by $\mathcal{P}_{uv}$ the set of all $u - v$ paths.

The second problem we have to deal with is that the value of the dual solution can be so small that we find a very small set of edges that already cover a large fraction of the multicommodity flow. However, we can use Lemma 6.37 to argue that a weight assignment on the edges that covers a large multicommodity flow in a (μ, λ)-connected set cannot have very small weight.

Lemma 6.39. *Let H be a hypergraph and let $0 < \lambda < 1/16$ be a constant. Then there is fractional independent set μ, a $(\mu, \lambda/6)$-connected set W, and a partition $(K_1, \ldots, K_k)$ of W such that $k = \Omega(\lambda\sqrt{\mathrm{con}_\lambda(H)})$, and for every $1 \le i \le k$, K_i is a clique with $\mu(K_i) \ge 1/2$.*

Proof. Let k be the largest integer such that $\mathrm{con}_\lambda(H) \ge 3T + 2k$ holds, where $T := (56/\lambda)^2 \cdot k^2$; it is clear that $k = \Omega(\lambda\sqrt{\mathrm{con}_\lambda(H)})$. Let μ_0 be a fractional independent set and W_0 be a (μ_0, λ)-connected set with $\mu_0(W_0) = \mathrm{con}_\lambda(H)$. We can assume that $\mu_0(v) > 0$ if and only if $v \in W_0$. This also implies that W_0 is in one connected component of H.

Highly loaded edges. First, we want to modify μ_0 such that there is no edge e with $\mu_0(e) \ge 1/2$. The following claim shows that we can achieve this by restricting μ_0 to an appropriate subset W of W_0.

Claim 6.40. *There is a subset $W \subseteq W_0$ such that $\mu_0(W) \ge \mathrm{con}_\lambda(H) - k$ and $\mu_0(e \cap W) < 1/2$ for every edge e.*

Proof. Let us choose edges g_1, g_2, ... as long as possible with the requirement $\mu_0(K_i) \ge 1/2$ for $K_i := (g_i \cap W_0) \setminus \bigcup_{j=1}^{i-1} K_j$. If we can select at least k such edges, then the cliques K_1, ..., K_k satisfy the requirements of Lemma 6.39 and we are done. Indeed, $W' := \bigcup_{i=1}^{k} K_i \subseteq W_0$ is a (μ_0, λ)-connected set, $\mu_0(K_i) \ge 1/2$, and $(K_1, \ldots, K_k)$ is a partition of W' into cliques.

Thus we can assume that the selection of the edges stops at edge g_t for some $t < k$. Let $W := W_0 \setminus \bigcup_{i=1}^{t} K_i$. Observe that there is no edge $e \in E(H)$ with $\mu_0(e \cap W) \ge 1/2$, as in this case the selection of the edges could be continued with $g_{t+1} := e$. Furthermore, we have $\mu_0(W) = \mu_0(W_0 \setminus \bigcup_{i=1}^{t} K_i) > \mu_0(W_0) - k = \mathrm{con}_\lambda(H) - k$, as required. □

Moderately connected pairs. Let us define μ such that $\mu(v) = 2\mu_0(v)$ if $v \in W$ and $\mu(v) = 0$ otherwise. By Claim 6.40, μ is a fractional independent set. The set W is (μ_0, λ)-connected (recall that a subset of (μ_0, λ)-connected is also (μ_0, λ)-connected). However, W is not necessarily (μ, λ)-connected. In the next step, we find a large collection of pairs (A_i, B_i) that violate (μ, λ)-connectivity. Informally, we can say that these pairs (A_i, B_i) are "moderately connected": denoting $w_i = \min\{\mu(A_i), \mu(B_i)\}$, the minimum value of a fractional (A_i, B_i)-separator for such a pair is less than λw_i (because the pair (A_i, B_i) violates (μ, λ)-connectivity), but at least $\lambda w_i/2 = \lambda \min\{\mu_0(A_i), \mu_0(B_i)\}$ (because W is (μ_0, λ)-connected).

Claim 6.41. *There are disjoint sets $A_1, B_1, \ldots, A_r, B_r \subseteq W$ such that for every $1 \leq i \leq r$ there is a fractional (A_i, B_i)-separator with weight less than λw_i for $w_i := \min\{\mu(A_i), \mu(B_i)\}$ and $w := \sum_{i=1}^{r} w_i \geq T$.*

Proof. Let us greedily select a maximal collection of pairs (A_1, B_1), $\ldots$, (A_r, B_r) with the property that there is a fractional (A_i, B_i)-separator with weight less than λw_i for $w_i := \min\{\mu(A_i), \mu(B_i)\}$. Note that every fractional separator has value at least 1 (as W is in a single component of H), thus $\lambda w_i > 1$ holds, implying $w_i > 1/\lambda > 1$. We can assume that $\mu(A_i), \mu(B_i) \leq w_i + 1$: if, say, $\mu(A_i) > \mu(B_i) + 1$, then removing an arbitrary vertex of A_i decreases $\mu(A_i)$ by at most one (as μ is a fractional independent set) without changing $\min\{\mu(A_i), \mu(B_i)\}$, hence there would be a smaller pair of sets with the required properties. Therefore, we have $2w_i \leq \mu(A_i \cup B_i) \leq 2w_i + 1 \leq 3w_i$ for every $1 \leq i \leq r$.

Suppose for contradiction that $w := \sum_{i=1}^{r} w_i < T$. Let $W' := W \setminus \bigcup_{i=1}^{r}(A_i \cup B_i)$. As $\mu(\bigcup_{i=1}^{r}(A_i \cup B_i)) \leq \sum_{i=1}^{r} 3w_i = 3w < 3T$, we have $\mu(W') > \mu(W) - 3T = 2\mu_0(W) - 3T \geq 2\,\mathrm{con}_\lambda(H) - 2k - 3T \geq \mathrm{con}_\lambda(H)$. Since the greedy selection stopped, there is no fractional (A', B')-separator of value less than $\lambda \cdot \min\{\mu(A'), \mu(B')\}$ for any disjoint $A', B' \subseteq W'$, that is, W' is (μ, λ)-connected with $\mu(W') > \mathrm{con}_\lambda(H)$, contradicting the definition of $\mathrm{con}_\lambda(H)$. ⌟

Finding a multicommodity flow. Let (A_1, B_1), $\ldots$, (A_r, B_r) be as in Claim 6.41. Since there is a fractional (A_i, B_i)-separator of value less than λw_i, the maximum value of a μ-demand multicommodity flow between pairs (A_1, B_1), $\ldots$, (A_r, B_r) is less than λw. Let y be an optimum dual solution; we give a lower bound on the total weight of the edge variables.

Claim 6.42. $\sum_{e \in E(H)} y(e) \geq 2k$.

Proof. Let $A := \bigcup_{i=1}^{r} A_i$ and $B := \bigcup_{i=1}^{r} B_i$. Let $A^* := \{u \in A \mid y(u) \leq 1/4\}$, $B^* := \{v \in B \mid y(v) \leq 1/4\}$, $A_i^* = A_i \cap A^*$, $B_i^* = B_i \cap B^*$, and $w_i^* = \min\{\mu(A_i^*), \mu(B_i^*)\}$. For each i, the value of w_i^* is either at least $w_i/2$, or less than that. Assume without loss of generality that there is a $1 \leq r^* \leq r$ such that $w_i^* \geq w_i/2$ if and only if $i \leq r^*$. Let $w^* = \sum_{i=1}^{r^*} w_i^*$.

We claim that $w^* \geq w/4$. Note that $w_i^* < w_i/2$ means that either $\mu(A_i^*) < w_i/2$ or $\mu(B_i^*) < w_i/2$; as $\mu(A_i), \mu(B_i) \geq w_i$, this is only possible if $\mu(A_i \setminus A^*) + \mu(B_i \setminus B^*) > w_i/2$. Suppose first that $\sum_{i=r^*+1}^{r} w_i > w/2$. This would imply

$$\mu((A \setminus A^*) \cup (B \setminus B^*)) \geq \sum_{i=r^*+1}^{r} (\mu(A_i \setminus A^*) + \mu(B_i \setminus B^*)) > \sum_{i=r^*+1}^{r} w_i/2 > w/4.$$

However, $y(u) > 1/4$ for every $u \in (A \setminus A^*) \cup (B \setminus B^*)$, thus $\sum_{v \in A \cup B} \mu(v) y(v) \geq \mu((A \setminus A^*) \cup (B \setminus B^*))/4 \geq w/16 > \lambda w$ (since $\lambda < 1/16$), a contradiction with the assumption that the optimum is at most λw. Thus we can assume that $\sum_{i=r^*+1}^{r} w_i \leq w/2$ and hence $\sum_{i=1}^{r^*} w_i \geq w/2$. Together with $w_i^* \geq w_i/2$ for every $1 \leq i \leq r^*$, this implies $w^* \geq w/4$.

As $y(a), y(b) \leq 1/4$ for every $a \in A_i^*$, $b \in B_i^*$, it is clear that for every $A_i^* - B_i^*$ path P, the total weight of the edges intersecting P has to be at least $1/2$ in assignment y. Therefore, if we define $y^* : E(H) \to \mathbb{R}^+$ by $y^*(e) = 2y(e)$ for every $e \in E(H)$, then y^* covers every $A_i^* - B_i^*$ path. Let $W^* = \bigcup_{i=1}^{r^*}(A_i^* \cup B_i^*)$. We use Lemma 6.37 for the (μ, λ)-connected set W^*, the pairs $(A_1^*, B_1^*), \ldots, (A_{r^*}^*, B_{r^*}^*)$, and for the weight assignment y^*. Note that $w_i^* \geq w_i/2 \geq 1/2$ for every i. It follows that the total weight of y^* on the edges is at least $(\lambda/7) \cdot \sqrt{w^*} \geq (\lambda/14) \cdot \sqrt{w}$, which means that $\sum_{e \in E(H)} y(e) \geq (\lambda/28) \cdot \sqrt{w} \geq (\lambda/28) \cdot \sqrt{T} \geq 2k$. □

Locating the cliques. Let y be an optimum dual solution for the maximum multicommodity flow problem with pairs $(A_1, B_1), \ldots, (A_r, B_r)$ and let flow F be the sum of the flows obtained from an optimum primal solution.

Claim 6.43. *There are k pairwise-disjoint cliques $K_1, \ldots, K_k$ and a set of k subflows $f_1, \ldots, f_k$ of F, each of them having value at least $1/2$, such that every path appearing in f_i intersects K_i and is disjoint from K_j for every $j \neq i$.*

Proof. Let $F^{(0)} = F$ and for $i = 1, 2, \ldots,$ let $F^{(i)}$ be the flow obtained from $F^{(0)}$ by removing $f_1, \ldots, f_i$. Let $c(e, F^{(i)})$ be the total weight of the paths in $F^{(i)}$ intersecting edge e and let $C_i = \sum_{e \in E(H)} y(e) c(e, F^{(i)})$. By complementary slackness, $c(e, F^{(0)}) = 1$ for each $e \in E(H)$ with $y(e) > 0$ and hence $C_0 = \sum_{e \in E(H)} y(e) \geq 2k$.

Let us select e_i to be an edge such that $c(e_i, F^{(i-1)})$ is maximum possible and let $K_i := e_i \setminus \bigcup_{j=1}^{i-1} e_j$. Let the flow f_i contain all the paths of $F^{(i-1)}$ intersecting e_i. Observe that the paths appearing in f_i do not intersect $e_1, \ldots, e_{i-1}$ (otherwise they would be in one of $f_1, \ldots, f_{i-1}$ and hence they would no longer be in $F^{(i-1)}$), thus clique K_i intersects every path in f_i.

For every $u - v$ path P appearing in $F^{(0)}$, we get $\sum_{e \in E(H), e \cap P \neq \emptyset} y(e) + y(u) + y(v) = 1$ from complementary slackness: if the primal variable corresponding to P is nonzero, then the corresponding dual constraint is tight. In particular, this means that the total weight of the edges intersecting such a path P is at most 1 in y. As $F^{(i-1)}$ is a subflow of $F^{(0)}$, this is also true for every path P in $F^{(i-1)}$. This means that when we remove a path of weight γ from $F^{(i-1)}$ to obtain $F^{(i)}$, then the total weight of the edges e for which $c(e, F^{(i-1)})$ decreases by γ is at most 1, i.e., C_{i-1} decreases by at most γ. As only the paths intersecting e_i are removed from $F^{(i-1)}$ and the total weight of the paths intersecting e_i is at most 1, we get that $C_i \geq C_{i-1} - 1$ and hence $C_i \geq C_0 - k \geq C_0/2$ for $i \leq k$. Since $C_0 = \sum_{e \in E(H)} y(e)$ and $C_i = \sum_{e \in E(H)} y(e) c(e, F^{(i)}) \geq C_0/2$, it follows that there has to be at least one edge e with $c(e, F^{(i)}) \geq 1/2$. Thus in each step, we can select an edge e_i such that that the total weight of the paths in $F^{(i)}$ intersecting e_i is at least $1/2$, and hence the value of f_i is at least $1/2$ for every $1 \leq i \leq k$. ⌟

Moving the highly connected set. Let $U = \bigcup_{i=1}^{k} K_i$.

Claim 6.44. *There is a fractional independent set μ' such that U is a $(\mu', \lambda/6)$-connected set with $\mu'(K_i) \geq 1/2$ for every $1 \leq i \leq r$.*

Proof. Each path P in f_i is a path with endpoints in W and intersecting K_i. Let us truncate each path P in f_i such that its first endpoint is still in W and its second endpoint is in K_i; let f_i' be the (W, K_i)-flow obtained by truncating every path in f_i. Note that f_i' is still a flow and the sum F' of $f_1', \ldots, f_k'$ is a (W, U)-flow. Let $\mu_1 = \mu$ and let $\mu_2(v)$ be the total weight of the paths in F' with second endpoint v. It is clear that μ_2 is a fractional independent set, $\mu_2(K_i) \geq 1/2$, and F is a (μ_1, μ_2)-demand (W, U)-flow with value $\mu_2(U)$. Thus by Lemma 6.38, U is a $(\mu_2, \lambda/6)$-connected set with the required properties. ⌟

Primal LP

$$\begin{aligned} &\text{maximize } \epsilon \\ &\text{s.t.} \\ &\sum_{1\le i<j\le k} \sum_{\substack{P\in\mathcal{P}_{i,j},\\ P\cap e\neq\emptyset}} x(P) \le 1 \quad \forall e \in E(H) \\ &\sum_{P\in\mathcal{P}_{i,j}} x(P) \ge \epsilon \quad \forall 1 \le i < j \le k \\ &x(P) \ge 0 \quad \begin{array}{l}\forall 1 \le i < j \le k,\\ P \in \mathcal{P}_{i,j}\end{array} \end{aligned}$$

Dual LP

$$\begin{aligned} &\text{minimize } \sum_{e\in e(H)} y(e) \\ &\text{s.t.} \\ &\sum_{e\in E(H), e\cap P\neq\emptyset} y(e) \ge \ell_{i,j} \quad \begin{array}{l}\forall 1 \le i < j \le k,\\ P \in \mathcal{P}_{i,j}\end{array} \\ &\sum_{1\le i<j\le k} \ell_{i,j} \ge 1 \\ &y(e) \ge 0 \quad \forall e \in E(H) \\ &\ell_{i,j} \ge 0 \quad \forall 1 \le i < j \le k \end{aligned}$$

Figure 6.6: Primal and dual linear programs for uniform concurrent flow on $W = (X_1, \ldots, X_k)$. We denote by $\mathcal{P}_{i,j}$ the set of all $X_i - X_j$ paths.

The set U, the partition $(K_1, \ldots, K_r)$, and the fractional independent set μ' clearly satisfy the requirements of the lemma. □

6.6.2 Concurrent flows and embedding

Let W be a set of vertices and let $(X_1, \ldots, X_k)$ be a partition of W. A *uniform concurrent flow of value ϵ on $(X_1, \ldots, X_k)$* is a compatible set of $\binom{k}{2}$ flows $F_{i,j}$ $(1 \le i < j \le k)$ where $F_{i,j}$ is an (X_i, X_j)-flow of value ϵ. The maximum value of a uniform concurrent flow on W can be expressed as the optimum values of the primal and dual linear programs in Figure 6.6. Intuitively, the dual linear program expresses that the "distance" of X_i and X_j is at least $\ell_{i,j}$ (where distance is measured as the minimum total weight of the edges intersected by an $X_i - X_j$ path) and the sum of these $\binom{k}{2}$ distances is at least 1.

If H is connected, then the maximum value of a uniform concurrent flow on $(X_1, \ldots, X_k)$ is at least $1/\binom{k}{2} = \Omega(k^{-2})$: if each of the $\binom{k}{2}$ flows has value $1/\binom{k}{2}$, then they are clearly compatible. The following lemma shows that in a (μ, λ)-connected set, if the sets $X_1, \ldots, X_k$ are cliques and $\mu(X_i) \ge 1/2$ for every i, then we can guarantee a better bound of $\Omega(k^{-\frac{3}{2}})$.

Lemma 6.45. *Let H be a hypergraph, μ a fractional independent set of H, and $W \subseteq V(H)$ a (μ, λ)-connected set for some $0 < \lambda < 1$. Let $(K_1, \ldots, K_k)$ (for some $k \ge 1$) be a partition of W such that K_i is a clique and $\mu(K_i) \ge 1/2$ for every $1 \le i \le k$. Then there is a uniform concurrent flow of value $\Omega(\lambda/k^{\frac{3}{2}})$ on $(K_1, \ldots, K_k)$.*

Proof. Suppose that there is no uniform concurrent flow of value $\beta \cdot \lambda/k^{\frac{3}{2}}$, where $\beta > 0$ is a sufficiently small universal constant specified later. This means that the dual linear program has a solution having value less than that. Let us fix such a solution $(y, \ell_{i,j})$ of the dual linear program. In the following, for every path P, we denote by $y(P) := \sum_{e\in E(H), e\cap P\neq\emptyset} y(e)$ the total weight of the edges intersecting P. It is clear from the dual linear program that $y(P) \ge \ell_{i,j}$ for every $P \in \mathcal{P}_{i,j}$.

We construct two graphs G_1 and G_2: the vertex set of both graphs is $\{1, \ldots, k\}$ and for every $1 \le i < j \le k$, vertices i and j are adjacent in G_1 (resp., G_2) if and only if $\ell_{i,j} > 1/(3k^2)$ (resp., $\ell_{i,j} > 1/k^2$). Note that G_2 is a subgraph of G_1. First we prove the following claim:

Claim 6.46. *If the distance of u and v is at most 3 in the* complement *of G_1, then u and v are not adjacent in G_2.*

Proof. Suppose that uw_1w_2v is a path of length 3 in the complement of G_1 (the same argument works for paths of length less than 3). By definition of G_1, there is a $K_u - K_{w_1}$ path P_1, a $K_{w_1} - K_{w_2}$ path P_2, and a $K_{w_2} - K_v$ path P_3 such that $y(P_1), y(P_2), y(P_3) \leq 1/(3k^2)$. Since K_{w_1} and K_{w_2} are cliques, paths P_1 and P_2 touch, and paths P_2 and P_3 touch. Thus by concatenating the three paths, we can obtain a $K_u - K_v$ path P with $y(P) \leq y(P_1) + y(P_2) + y(P_3) \leq 1/k^2$, implying that u and v are not adjacent in G_2, proving the claim. Note that the proof of this claim is the only point where we use that the K_i's are cliques. ⌟

Let $y' : E(H) \to \mathbb{R}^+$ be defined by $y'(e) := 3k^2 \cdot y(e)$, thus y' has total weight less than $3\beta \cdot \lambda\sqrt{k}$. Suppose first that G_1 has a matching $a_1b_1, \ldots, a_mb_m$ of size $m = \lceil k/4 \rceil$. This means that y' covers every $K_{a_i} - K_{b_i}$ path for every $1 \leq i \leq \lceil k/4 \rceil$. Therefore, by Lemma 6.37, y' has weight at least $(\lambda/7) \cdot \sqrt{\lceil k/4 \rceil \cdot (1/2)} > 3\beta \cdot \lambda\sqrt{k}$, if β is sufficiently small, yielding a contradiction.

Thus the size of the maximum matching in G_1 is less than $\lceil k/4 \rceil$, which means that there is a vertex cover S_1 of size at most $k/2$. Let $S_2 \subseteq S_1$ contain those vertices of S_1 that are adjacent to every vertex outside S_1 in G_1. We claim that S_2 is a vertex cover of G_2. Suppose that there is an edge uv of G_2 for some $u, v \notin S_2$. By the definition of S_2, either $u \notin S_1$, or there is a vertex $w_1 \notin S_1$ such that u and w_1 are not adjacent in G_1. Similarly, either v is not in S_1, or it is not adjacent in G_1 to some $w_2 \notin S_1$. Since vertices not in S_1 are not adjacent in G_1 (as S_1 is a vertex cover of G_1), we get that the distance of u and v is at most 3 in the complement of G_1. Thus by the claim, u and v are not adjacent in G_2.

Let us give an upper bound on $\sum_{1 \leq i < j \leq k} \ell_{i,j}$ by bounding $\ell_{i,j}$ separately for pairs that are adjacent in G_2 and for pairs that are not adjacent in G_2. The number of edges in G_2 is at most $|S_2|k$ (as S_2 is vertex cover). The total weight of y, which is less than $\beta \cdot \lambda/k^{\frac{3}{2}}$, is an upper bound on any $\ell_{i,j}$. Furthermore, if i and j are not adjacent in G_2, then we have $\ell_{i,j} \leq 1/k^2$. Therefore,

$$1 \leq \sum_{1 \leq i < j \leq k} \ell_{i,j} \leq |S_2|k \cdot \beta \cdot \lambda/k^{\frac{3}{2}} + \binom{k}{2}(1/k^2) \leq \beta \cdot \lambda |S_2|/\sqrt{k} + 1/2,$$

which implies that $|S_2| \geq \sqrt{k}/(2\beta\lambda)$. Let $A := \bigcup_{i \in S_2} K_i$ and $B := \bigcup_{i \notin S_1} K_i$; we have $\mu(A) \geq |S_2| \cdot (1/2) \geq \sqrt{k}/(4\beta\lambda)$ and $\mu(B) \geq (1/2) \cdot (k - |S_1|)) \geq k/4$. As every vertex of S_2 is adjacent in G_1 with every vertex outside S_1, assignment y' covers every $A - B$ path. However, y' has weight less than $3\beta \cdot \lambda\sqrt{k} < \min\{\sqrt{k}/(4\beta\lambda), k/4\}$ (using that $\lambda \leq 1$ and assuming that β is sufficiently small), contradicting the assumption that W is (μ, λ)-connected. □

Intuitively, the intersection structure of the paths appearing in a uniform concurrent flow on cliques $K_1, \ldots, K_k$ is reminiscent of the edges of the complete graph on k vertices: if $\{i_1, j_1\} \cap \{i_2, j_2\} \neq \emptyset$, then every path of F_{i_1,j_1} touches every path of F_{i_2,j_2}. We use the following result from [185], which shows that the line graph of cliques have good embedding properties. If G is a graph and $q \geq 1$ is an integer, then the *blow up* $G^{(q)}$ is obtained from G by replacing every vertex v with a clique K_v of size q and for every edge uv of G, connecting every vertex of the clique K_u with every vertex of the clique K_v. Let L_k be the line graph of the complete graph on k vertices. We recall the following lemma from Section 3.2.2.

Lemma 6.47 (Restated Lemma 3.13)**.** *For every $k > 1$ there is a constant $n_k > 0$ such that for every G with $|E(G)| > n_k$ and no isolated vertices, the graph G is a minor of $L_k^{(q)}$ for $q = \lceil 130|E(G)|/k^2 \rceil$. Furthermore, a minor mapping can be found in time polynomial in the size of G.*

Using the terminology of embeddings, a minor mapping of G into $L_k^{(q)}$ can be considered as an embedding from G to L_k where every vertex of L_k appears in the image of at most q vertices, i.e., the vertex depth of the embedding is at most q. Thus we can restate Lemma 6.47 the following way:

Lemma 6.48. *For every $k > 1$ there is a constant $n_k > 0$ such that for every G with $|E(G)| > n_k$ and no isolated vertices, the graph G has an embedding into L_k with vertex depth $O(|E(G)|/k^2)$. Furthermore, such an embedding can be found in time polynomial in the size of G.*

Now we are ready to prove Theorem 6.35, the main result of the section:

Proof (of Theorem 6.35). By Lemma 6.39 and Lemma 6.45, for some $k = \Omega(\lambda\sqrt{\mathrm{con}_\lambda(H)})$, there are cliques $K_1, \ldots, K_k$ and a uniform concurrent flow $F_{i,j}$ $(1 \leq i < j \leq k)$ of value $\epsilon = \Omega(\lambda/k^{\frac{3}{2}})$ on $(K_1, \ldots, K_k)$. By trying all possibilities for the cliques and then solving the uniform concurrent flow linear program, we can find these flows (the time required for this step is a constant $f(H, \lambda)$ depending only on H and λ) . Let w_0 be the smallest positive weight appearing in the flows.

Let $m = |E(G)|$ and suppose that $m \geq n_k$, for the constant n_k in Lemma 6.48. Thus the algorithm of Lemma 6.48 can be used to find a an embedding ψ from G to L_k with vertex depth $q = O(m/k^2)$. Let us denote by $v_{\{i,j\}}$ $(1 \leq i < j \leq k)$ the vertices of L_k with the meaning that distinct vertices $v_{\{i_1,j_1\}}$ and $v_{\{i_2,j_2\}}$ are adjacent if and only if $\{i_1, j_1\} \cap \{i_2, j_2\} \neq \emptyset$.

We construct an embedding φ from G to H the following way. The set $\varphi(u) \subseteq V(H)$ is obtained from the set $\psi(u) \subseteq V(L_k)$ by replacing each vertex of $v_{\{i,j\}} \in \psi(u) \subseteq V(L_k)$ by a path from the flow $F_{i,j}$ (thus $\varphi(u)$ is the union of $|\psi(u)|$ paths of H). We select the paths in such a way that the following requirement is satisfied: a path P of $F_{i,j}$ having weight w is selected into the image of at most $\lceil (q/\epsilon) \cdot w \rceil$ vertices of G. We set $m_{H,\lambda}$ sufficiently large that $(q/\epsilon) \cdot w_0 \geq 1$ (note that q depends on m, but ϵ and w_0 depends only on H and λ). Thus if $m \geq m_{H,\lambda}$, then $\lceil (q/\epsilon) \cdot w \rceil \leq 2(q/\epsilon) \cdot w$. Since the total weight of the paths in $F_{i,j}$ is ϵ, these paths can accommodate the image of at least $(q/\epsilon) \cdot \epsilon = q$ vertices. As each vertex $v_{\{i,j\}}$ of L_k appears in the image of at most q vertices of G in the mapping ψ, we can satisfy the requirement.

It is easy to see that if u_1 and u_2 are adjacent in G, then $\varphi(u_1)$ and $\varphi(u_2)$ touch: in this case, there are vertices $v_{\{i_1,j_1\}} \in \psi(u_1)$, $v_{\{i_2,j_2\}} \in \psi(u_2)$ that are adjacent or the same in L_k (that is, there is a $t \in \{i_1, j_1\} \cap \{i_2, j_2\}$), and the corresponding paths of F_{i_1,j_1} and F_{i_2,j_2} selected into $\varphi(u_1)$ and $\varphi(u_2)$ touch, as they both intersect the clique K_t. With a similar argument, we can show that $\varphi(u)$ is connected.

To bound the edge depth of the embedding φ, consider an edge e. The total weight of the paths intersecting e is at most 1 and a path with weight w is used in the image of at most $2(q/\epsilon) \cdot w$ vertices. Each path intersects e in at most 2 vertices (as we can assume that the paths appearing in the flows are minimal), thus a path with weight w contributes at most $4(q/\epsilon) \cdot w$ to the depth of e. Thus the edge depth of φ is at most $4(q/\epsilon) = O(m/(\lambda\sqrt{k})) = O(m/(\lambda^{\frac{3}{2}} \mathrm{con}_\lambda(H)^{\frac{1}{4}}))$. □

6.7 From embeddings to hardness of CSP

We prove the main hardness result of the chapter in this section:

Theorem 6.49. *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs with unbounded submodular width. If there is an algorithm $\mathbb{A}$ and a function f such that $\mathbb{A}$ solves every instance I of $\mathrm{CSP}(\mathcal{H})$ with hypergraph $H \in \mathcal{H}$ in time $f(H) \cdot \|I\|^{o(\mathrm{subw}(H)^{1/4})}$, then the Exponential Time Hypothesis fails.*

In particular, Theorem 6.49 implies that $\mathrm{CSP}(\mathcal{H})$ for such a $\mathcal{H}$ is not fixed-parameter tractable:

Corollary 6.50. *If $\mathcal{H}$ is a recursively enumerable class of hypergraphs with unbounded submodular width, then $\mathrm{CSP}(\mathcal{H})$ is not fixed-parameter tractable, unless the Exponential Time Hypothesis fails.*

To prove Theorem 6.49, we show that a subexponential-time algorithm for 3SAT exists if CSP($\mathcal{H}$) can be solved "too fast" for some $\mathcal{H}$ with unbounded submodular width. We use the characterization of submodular width from Section 6.5 and the embedding results of Section 6.6 to reduce 3SAT to CSP($\mathcal{H}$) by embedding the incidence graph of a 3SAT formula into a hypergraph $H \in \mathcal{H}$. The basic idea of the proof is that if the 3SAT formula has m clauses and the edge depth of the embedding is m/r, then we can gain a factor r in the exponent of the running time. If submodular width is unbounded in $\mathcal{H}$, then we can make this gap r between the number of clauses and the edge depth arbitrary large, and hence the exponent can be arbitrarily smaller than the number of clauses, i.e., the algorithm is subexponential in the number of clauses.

Next we show that an embedding from graph G to hypergraph H can be used to simulate a binary CSP instance I_1 having primal graph G by a CSP instance I_2 whose hypergraph is H. The domain size and the size of the constraint relations of I_2 can grow very large in this transformation: the edge depth of the embedding determines how large this increase is.

Lemma 6.51. *Let $I_1 = (V_1, D_1, C_1)$ be a binary CSP instance with primal graph G and let φ be an embedding of G into a hypergraph H with edge depth q. Given I_1, H, and the embedding φ, it is possible to construct (in time polynomial in the size of the* output*) an equivalent CSP instance $I_2 = (V_2, D_2, C_2)$ with hypergraph H where the size of every constraint relation is at most $|D_1|^q$.*

Proof. For every $v \in V(H)$, let $U_v := \{u \in V(G) \mid v \in \varphi(u)\}$ be the set of vertices in G whose images contain v, and for every $e \in E(H)$, let $U_e := \bigcup_{v \in e} U_v$. Observe that for every $e \in E(H)$, we have $|U_e| \leq \sum_{v \in e} |U_v| \leq q$, since the edge depth of φ is q. Let D_2 be the set of integers between 1 and $|D_1|^q$. For every $v \in V(H)$, the number of assignments from U_v to D_1 is clearly $|D_1|^{|U_v|} \leq |D_1|^q$. Let us fix a bijection h_v between these assignments on U_v and the set $\{1, \ldots, |D_1|^{|U_v|}\} \subseteq D_2$.

The set C_2 of constraints of I_2 are constructed as follows. For each $e \in E(H)$, there is a constraint $\langle s_e, R_e \rangle$ in C_2, where s_e is an $|e|$-tuple containing an arbitrary ordering of the elements of e. The relation R_e is defined the following way. Suppose that v_i is the i-th coordinate of s_e and consider a tuple $t = (d_1, \ldots, d_{|e|}) \in D_2^{|e|}$ of integers where $1 \leq d_i \leq |D_1|^{|U_{v_i}|}$ for every $1 \leq i \leq |e|$. This means that d_i is in the image of h_{v_i} and hence $f_i := h_{v_i}^{-1}(d_i)$ is an assignment from U_{v_i} to D_1. We define relation R_e such that it contains tuple t if the following two conditions hold. First, we require that the assignments $f_1, \ldots, f_{|e|}$ are *consistent* in the sense that $f_i(u) = f_j(u)$ for any i, j and $u \in U_{v_i} \cap U_{v_j}$. In this case, $f_1, \ldots, f_{|e|}$ together define an assignment f on $\bigcup_{i=1}^{|e|} U_{v_i} = U_e$. The second requirement is that this assignment f satisfies every constraint of I_1 whose scope is contained in U_e, that is, for every constraint $\langle (u_1, u_2), R \rangle \in C_1$ with $\{u_1, u_2\} \subseteq U_e$, we have $(f(u_1), f(u_2)) \in R$. This completes the description of the instance I_2.

Let us bound the maximum size of a relation of I_2. Consider the relation R_e constructed in the previous paragraph. It contains tuples $(d_1, \ldots, d_{|e|}) \in D_2^{|e|}$ where $1 \leq d_i \leq |D_1|^{|U_{v_i}|}$ for every $1 \leq i \leq |e|$. This means that

$$|R_e| \leq \prod_{i=1}^{|e|} |D_1|^{|U_{v_i}|} = |D_1|^{\sum_{i=1}^{|e|} |U_{v_i}|} \leq |D_1|^q, \tag{6.4}$$

where the last inequality follows from the fact that φ has edge depth at most q.

To prove that I_1 and I_2 are equivalent, assume first that I_1 has a solution $f_1 : V_1 \to D_1$. For every $v \in V_2$, let us define $f_2(v) := h_v(\mathrm{pr}_{U_v} f_2)$, that is, the integer between 1 and $|D_1|^{|U_v|}$ corresponding to the projection of assignment f_2 to U_v. It is easy to see that f_2 is a solution of I_2.

Assume now that I_2 has a solution $f_2 : V_2 \to D_2$. For every $v \in V(H)$, let $f_v := h_v^{-1}(f_2(v))$ be the assignment from U_v to D_1 that corresponds to $f_2(v)$ (note that by construction, $f_2(v)$ is at

most $|D_1|^{|U_v|}$, hence $h_v^{-1}(f_2(v))$ is well-defined). We claim that these assignments are compatible: if $u \in U_{v'} \cap U_{v''}$ for some $u \in V(G)$ and $v', v'' \in V(H)$, then $f_{v'}(u) = f_{v''}(u)$. Recall that $\varphi(u)$ is a connected set in H, hence there is a path between v' and v'' in $\varphi(u)$. We prove the claim by induction on the distance between v' and v'' in $\varphi(u)$. If the distance is 0, that is, $v' = v''$, then the statement is trivial. Suppose now that the distance of v' and v'' is $d > 0$. This means that v' has a neighbor $z \in \varphi(u)$ such that the distance of z and v'' is $d - 1$. Therefore, $f_z(u) = f_{v''}(u)$ by the induction hypothesis. Since v' and z are adjacent in H, there is an edge $E \in E(H)$ containing both v' and z. From the way I_2 is defined, this means that $f_{v'}$ and f_z are compatible and $f_{v'}(u) = f_z(u) = f_{v''}(u)$ follows, proving the claim. Thus the assignments $\{f_v \mid v \in V(H)\}$ are compatible and these assignments together define an assignment $f_1 : V(G) \to D$. We claim that f_1 is a solution of I_1. Let $c = \langle (u_1, u_2), R \rangle$ be an arbitrary constraint of I_1. Since $u_1u_2 \in E(G)$, sets $\varphi(u_1)$ and $\varphi(u_2)$ touch, thus there is an edge $e \in E(H)$ that contains a vertex $v_1 \in \varphi(u_1)$ and a vertex $v_2 \in \varphi(u_2)$ (or, in other words, $u_1 \in U_{v_1}$ and $u_2 \in U_{v_2}$). The definition of c_e in I_2 ensures that f_1 restricted to $U_{v_1} \cup U_{v_2}$ satisfies every constraint of I_1 whose scope is contained in $U_{v_1} \cup U_{v_2}$; in particular, f_1 satisfies constraint c. □

Now we are ready to prove Theorem 6.49, the main result of the section. We show that if there is a class $\mathcal{H}$ of hypergraphs with unbounded submodular width such that CSP($\mathcal{H}$) is FPT, then this algorithm can be used to solve 3SAT in subexponential time. The main ingredients are the embedding result of Theorem 6.35, and Lemmas 3.14 and 6.51 above on reduction to CSP. Furthermore, we need a way of choosing an appropriate hypergraph from the set $\mathcal{H}$. As discussed earlier, the larger the submodular width of the hypergraph is, the more we gain in the running time. However, we should not spend too much time on constructing the hypergraph and on finding an embedding. Therefore, we use the same technique as in Chapter 3: we enumerate a certain number of hypergraphs and we try all of them simultaneously. The number of hypergraphs enumerated depends on the size of the 3SAT instance. This will be done in such a way that guarantees that we do not spend too much time on the enumeration, but eventually every hypergraph in $\mathcal{H}$ is considered for sufficiently large input sizes.

Proof (of Theorem 6.49). Let us fix a $\lambda > 0$ that is sufficiently small for Theorems 6.21 and 6.35. Suppose that there is an $f_1(H)n^{o(\mathrm{subw}(H)^{1/4})}$ time algorithm $\mathbb{A}$ for CSP($\mathcal{H}$). We can express the running time as $f_1(H)n^{\mathrm{subw}(H)^{1/4}/\iota(\mathrm{subw}(H))}$ for some unbounded nondecreasing function ι with $\iota(1) > 0$. We construct an algorithm $\mathbb{B}$ that solves 3SAT in subexponential time by using algorithm $\mathbb{A}$ as subroutine.

Given an instance I of 3SAT with n variables and m clauses and a hypergraph $H \in \mathcal{H}$, we can solve I the following way. First we use Lemma 3.14 to transform I into a CSP instance $I_1 = (V_1, D_1, C_1)$ with $|V_1| = n + m$, $|D_1| = 3$, and $|C_1| = 3m$. Let G be the primal graph of I_1, which is a graph having $3m$ edges. It can be assumed that m is greater than some constant $m_{H,\lambda}$ of Theorem 6.35, otherwise the instance can be solved in constant time. Therefore, the algorithm of Theorem 6.35 can be used to find an embedding φ of G into H with edge depth $q = O(m/(\lambda^{\frac{3}{2}} \mathrm{con}_\lambda(H)^{1/4}))$; by Theorem 6.21, we have that $\mathrm{con}_\lambda(H) = \Omega(\mathrm{subw}(H))$ and hence $q \leq c_\lambda m/\mathrm{subw}(H)^{1/4}$ for some constant c_λ depending only on λ. By Lemma 6.51, we can construct an equivalent instance $I_2 = (V_2, D_2, C_2)$ whose hypergraph is H. By solving I_2 using the assumed algorithm $\mathbb{A}$ for CSP($\mathcal{H}$), we can answer if I_1 has a solution, or equivalently, if the 3SAT instance I has a solution.

We will call "running algorithm $\mathbb{A}[I, H]$" this way of solving the 3SAT instance I. Let us determine the running time of $\mathbb{A}[I, H]$. The two dominating terms are the time required to find embedding φ using the $f(H, \lambda)m^{O(1)}$ time algorithm of Theorem 6.49 and the time required to run $\mathbb{A}$ on I_2. The

size of every constraint relation in I_2 is at most $|D_1|^q = 3^q$, hence $\|I_2\| = O((|E(H)| + |V(H)|)3^q)$. Let $k = \mathrm{subw}(H)$. The total running time of $\mathbb{A}[I, H]$ can be bounded by

$$f(H,\lambda)m^{O(1)} + f_1(H)\|I_2\|^{k^{1/4}/\iota(k)} = f(H,\lambda)m^{O(1)} + f_1(H)(|E(H)| + |V(H|)^{k^{1/4}/\iota(k)} \cdot 3^{q\cdot k^{1/4}/\iota(k)}$$
$$= f_2(H,\lambda) \cdot m^{O(1)} \cdot 3^{c_\lambda m/\iota(k)}$$

for an appropriate function $f_2(H, \lambda)$ depending only on H and λ.

Algorithm $\mathbb{B}$ for 3SAT proceeds as follows. Let us fix an arbitrary computable enumeration H_1, H_2, ... of the hypergraphs in $\mathcal{H}$. Given an m-clause 3SAT formula I, algorithm $\mathbb{B}$ spends the first m steps on enumerating these hypergraphs; let H_ℓ be the last hypergraph produced by this enumeration (we assume that m is sufficiently large that $\ell \geq 1$). Next we start simulating the algorithms $\mathbb{A}[I, H_1]$, $\mathbb{A}[I, H_2]$, ..., $\mathbb{A}[I, H_\ell]$ in *parallel*. When one of the simulations stops and returns an answer, then we stop all the simulations and return the answer. It is clear that algorithm $\mathbb{B}$ will correctly decide the satisfiability of I.

We claim that there is a universal constant d such that for every s, there is an m_s such that for every $m > m_s$, the running time of $\mathbb{B}$ is at most $(m \cdot 2^{m/s})^d$ on an m-clause formula. Clearly, this means that the running time of $\mathbb{B}$ is $2^{o(m)}$.

For any positive integer s, let k_s be the smallest positive integer such that $\iota(k_s) \geq s$ (as ι is unbounded, this is well defined). Let i_s be the smallest positive integer such that $\mathrm{subw}(H_{i_s}) \geq k_s$ (as $\mathcal{H}$ has unbounded submodular width, this is also well defined). Set m_s sufficiently large that $m_s \geq f_2(H_{i_s}, \lambda)$ and the fixed enumeration of $\mathcal{H}$ reaches H_{i_s} in less then m_s steps. This means that if we run $\mathbb{B}$ on a 3SAT formula I with $m \geq m_s$ clauses, then $\ell \geq i_s$ and hence $\mathbb{A}[I, H_{i_s}]$ will be one of the ℓ simulations started by $\mathbb{B}$. The simulation of $\mathbb{A}[I, H_{i_s}]$ terminates in

$$f_2(H_{i_s},\lambda)m^{O(1)} \cdot 3^{c_\lambda m/\iota(\mathrm{subw}(H_{i_s}))} \leq m \cdot m^{O(1)} \cdot 3^{c_\lambda m/s}$$

steps. Taking into account that we simulate $\ell \leq m$ algorithms in parallel and all the simulations are stopped not later than the termination of $\mathbb{A}[I, H_{i_s}]$, the running time of $\mathbb{B}$ can be bounded polynomially by the running time of $\mathbb{A}[I, H_{i_s}]$. Therefore, there is a constant d such that the running time of $\mathbb{B}$ is at most $(m \cdot 2^{m/s})^d$, as required. □

Remark 6.52. Recall that if φ is an embedding of G into H, then the depth of an edge $e \in E(H)$ is $d_\varphi(e) = \sum_{v\in V(G)} |\varphi(v) \cap e|$. A variant of this definition would be to define the depth of e as $d'_\varphi(e) = |\{v \in V(G) \mid \varphi(v) \cap e \neq \emptyset\}|$, i.e., if $\varphi(v)$ intersects e, then v contributes only 1 to the depth of e, not $|\varphi(v) \cap e|$ as in the original definition. Let us call this variant *weak edge depth,* it is clear that the weak edge depth of an embedding is at most the edge depth of the embedding.

Lemma 6.51 can be made stronger by requiring only that the weak edge depth is at most q. Indeed, the only place where we use the bound on edge depth is in Inequality (6.4). However, the size of the relation R_e can be bounded by the number of possible assignments on U_e in instance I_1. If weak edge depth is at most q, then $|U_e| \leq q$, and the $|D_1|^q$ bound on the size of R_e follows.

Remark 6.53. A different version of CSP was investigated in [187], where each variable has a different domain, and each constraint relation is represented by a full truth table (see the exact definition in [187]). Let us denote by $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ this variant of the problem. It is easy to see that $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ can be reduced to $\mathrm{CSP}(\mathcal{H})$ in polynomial time, but a reduction in the other direction can possibly increase the representation of a constraint by an exponential factor. Nevertheless, the hardness results of this section apply to the "easier" problem $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ as well. What we have to verify is that the proof of Lemma 6.51 works even if I_2 is an instance of $\mathrm{CSP}_{\mathrm{tt}}$, i.e., the constraint relations have to be represented by truth tables. Inspection of the proof shows that it indeed

works: the product in Inequality (6.4) is exactly the size of the truth table describing the constraint corresponding to edge e, thus the $|D_1|^q$ upper bound remains valid even if constraints are represented by truth tables. Therefore, the hardness results of [187] are subsumed by the following corollary:

Corollary 6.54. *If $\mathcal{H}$ is a recursively enumerable class of hypergraphs with unbounded submodular width, then $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ is not fixed-parameter tractable, unless the Exponential Time Hypothesis fails.*

6.8 Conclusions

The main result of this chapter is introducing submodular width and proving that bounded submodular width is the property that determines the fixed-parameter tractability of $\mathrm{CSP}(\mathcal{H})$. The hardness result is proved assuming the Exponential Time Hypothesis. This conjecture was formulated relatively recently [146], but it turned out to be very useful in proving lower bounds in a variety of settings [16, 181, 185, 202].

Let us briefly review the main ideas that were necessary for proving the main result of the chapter:

- Recognizing that submodular width is the right property characterizing the complexity of the problem.
- A CSP instance can be partitioned into a bounded number of uniform instances (Section 6.4.2).
- The number of solutions in a uniform CSP instance can be described by a submodular function (Section 6.4.3).
- There is a connection between fractional separation and finding a separator minimizing an edge-dominated submodular cost function (Section 6.5.2).
- The transformation that turns b into b^*, and the properties of b^* that are more suitable than b for recursively constructing a tree decomposition (Section 6.5.1).
- Our results on fractional separation and the standard framework of finding tree decompositions show that large submodular width implies that there is a highly connected set (Section 6.5.3).
- A highly connected set can be turned into a highly connected set that is partitioned into cliques in an appropriate way (Section 6.6.1).
- A highly connected set with appropriate cliques implies that there is a uniform concurrent flow of large value between the cliques (Section 6.6.2).
- Similarly to [185], we use the observation that a concurrent flow is analogous to a line graph of a clique, hence it has good embedding properties (Section 6.6.2).
- Similarly to [185], an embedding in a hypergraph gives a way of simulating 3SAT with $\mathrm{CSP}(\mathcal{H})$ (Section 6.7).

An obvious question for further research is whether it is possible to prove a similar dichotomy result with respect to polynomial-time solvability. At this point, it is hard to see what the answer could be if we investigate the same question using the more restricted notion of polynomial time solvability. We know that bounded fractional hypertree width implies polynomial-time solvability (Chapter 5) and Theorem 6.49 shows that unbounded submodular width implies that the problem is not polynomial-time solvable (as it is not even fixed-parameter tractable). So only those classes of

hypergraphs are in the "gray zone" that have bounded submodular width but unbounded fractional hypertree width.

What could be the truth in this gray zone? A first possibility is that CSP($\mathcal{H}$) is polynomial-time solvable for every such class, i.e., Theorem 6.9 can be improved from fixed-parameter tractability to polynomial-time solvability. However, Theorem 6.9 uses the power of fixed-parameter tractability in an essential way (splitting into a double-exponential number of uniform instances), so it is not clear how such improvement is possible. A second possibility is that unbounded fractional hypertree width implies that CSP($\mathcal{H}$) is not polynomial-time solvable. Substantially new techniques would be required for such a hardness proof. The hardness proofs of this chapter and of [123, 185] are based on showing that a large problem space can be efficiently embedded into an instance with a particular hypergraph. However, the fixed-parameter tractability results show that no such embedding is possible in case of classes with bounded submodular width. Therefore, a possible hardness proof should embed a problem space that is comparable (in some sense) with the size of the hypergraph and should create instances where the domain size is bounded by a function of the size of the hypergraph. A third possibility is that the boundary of polynomial-time solvability is somewhere between bounded fractional hypertree width and bounded submodular width. Currently, there is no natural candidate for a property that could correspond to this boundary and, again, the hardness part of the characterization should be substantially different than what was done before. Finally, there is a fourth possibility: the boundary of the polynomial-time cases cannot be elegantly characterized by a simple combinatorial property. In general, if we consider the restriction of a problem to all possible classes of (hyper)graphs, then there is no a priori reason why an elegant characterization should exist that describes the easy and hard classes. For example, it is highly unlikely that there is an elegant characterization of those classes of graphs where solving the Maximum Independent Set problem is polynomial-time solvable. As discussed earlier, the fixed-parameter tractability of CSP($\mathcal{H}$) is a more robust question than its polynomial-time solvability, hence it is very well possible that only the former question has an elegant answer.

Bibliography

[1] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, pages 59–78, 2015.

→ *p. 8*

[2] A. Abboud, K. Bringmann, H. Dell, and J. Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In I. Diakonikolas, D. Kempe, and M. Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018), Los Angeles, CA, USA, June 25-29, 2018*, pages 253–266. ACM, 2018.

→ *p. 8*

[3] A. Abboud, V. V. Williams, and H. Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018.

→ *p. 8*

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

→ *p. 66*

[5] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007.

→ *p. 9*

[6] I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47:275–296, 2004.

→ *p. 78*

[7] I. Adler. *Width functions for hypertree decompositions*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2006.

→ *pp. 99 and 111*

[8] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *European J. Combin.*, 28(8):2167–2181, 2007.

→ *pp. 15, 60, 78, 82, 85, 100, and 110*

[9] I. Adler, S. G. Kolliopoulos, P. K. Krause, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Irrelevant vertices for the planar disjoint paths problem. *J. Comb. Theory, Ser. B*, 122:815–843,

2017.

→ *p. 19*

[10] A. Agarwal, N. Alon, and M. Charikar. Improved approximation for directed cut problems. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 671–680. ACM, New York, 2007.

→ *p. 120*

[11] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002)*, volume 2286 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2002.

→ *pp. 19 and 20*

[12] N. Alon, I. Newman, A. Shen, G. Tardos, and N. Vereshchagin. Partitioning multi-dimensional sets in a small number of "uniform" parts. *European J. Combin.*, 28(1):134–144, 2007.

→ *pp. 94 and 105*

[13] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

→ *pp. 9 and 47*

[14] O. Amini, F. Mazoit, N. Nisse, and S. ThomassÃl'. Submodular partition functions. *Discrete Mathematics*, 309(20):6000 – 6008, 2009.

→ *p. 102*

[15] E. Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

→ *p. 11*

[16] A. Andoni, P. Indyk, and M. Patrascu. On the optimality of the dimensionality reduction method. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 449–458, Washington, DC, USA, 2006. IEEE Computer Society.

→ *pp. 8 and 132*

[17] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

→ *pp. 9 and 11*

[18] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

→ *p. 7*

[19] A. Atserias, A. A. Bulatov, and V. Dalmau. On the power of k -consistency. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2007.

→ *p. 104*

[20] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 739–748, 2008.

→ *pp. 18 and 59*

[21] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

→ *pp. 18 and 59*

[22] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

→ *p. 8*

[23] G. Bagan, A. Durand, E. Filiot, and O. Gauwin. Efficient enumeration for conjunctive queries over x-underbar structures. In A. Dawar and H. Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2010.

→ *p. 17*

[24] M. Bateni, M. T. Hajiaghayi, and D. Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21:1–21:37, 2011.

→ *p. 19*

[25] C. Beeri, R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman, and M. Yannakakis. Properties of acyclic database schemes. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981)*, pages 355–362. ACM, 1981.

→ *pp. 17 and 93*

[26] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. Assoc. Comput. Mach.*, 30(3):479–513, 1983.

→ *pp. 17 and 93*

[27] C. Berge. *Graphs and Hypergraphs*. North Holland, 1976.

→ *pp. 15 and 60*

[28] U. Bertelè and F. Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

→ *p. 5*

[29] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM on Symposium on Theory of Computing (STOC 2007)*, pages 67–74, 2007.

→ *pp. 11 and 20*

[30] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993.

→ *p. 11*

[31] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

→ *p. 11*

[32] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997, 22nd International Symposium (MFCS 1997)*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1997.

→ *p. 9*

[33] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.

→ *p. 11*

[34] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

→ *pp. 11 and 20*

[35] É. Bonnet, P. Giannopoulos, and M. Lampis. On the parameterized complexity of red-blue points separation. In D. Lokshtanov and N. Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

→ *pp. 14 and 48*

[36] É. Bonnet and T. Miltzow. Parameterized hardness of art gallery problems. In P. Sankowski and C. D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

→ *pp. 14 and 48*

[37] É. Bonnet and F. Sikora. The graph motif problem parameterized by the structure of the input graph. *Discrete Applied Mathematics*, 231:78–94, 2017.

→ *pp. 14 and 48*

[38] G. Borradaile and H. Le. Optimal dynamic program for r-domination problems over tree decompositions. In J. Guo and D. Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

→ *pp. 8, 11, 20, and 21*

[39] K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2014), Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014.

→ *p. 8*

[40] K. Bringmann, P. Gawrychowski, S. Mozes, and O. Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1190–1206. SIAM, 2018.

→ *p. 8*

[41] K. Bringmann, D. Hermelin, M. Mnich, and E. J. van Leeuwen. Parameterized complexity dichotomy for Steiner multicut. *J. Comput. Syst. Sci.*, 82(6):1020–1043, 2016.

→ *p. 10*

[42] K. Bringmann, L. Kozma, S. Moran, and N. S. Narayanaswamy. Hitting set for hypergraphs of low vc-dimension. In P. Sankowski and C. D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

→ *pp. 14 and 48*

[43] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015), Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015.

→ *p. 8*

[44] K. Bringmann and M. Künnemann. Multivariate fine-grained complexity of longest common subsequence. In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1216–1235. SIAM, 2018.

→ *p. 8*

[45] A. A. Bulatov. Tractable conservative constraint satisfaction problems. In *18th Annual IEEE Symposium on Logic in Computer Science ((LICS 2003))*, page 321, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

→ *p. 14*

[46] A. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, 2006.

→ *p. 14*

[47] A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In C. Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017), Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017.

→ *p. 14*

[48] A. A. Bulatov, V. Dalmau, M. Grohe, and D. Marx. Enumerating homomorphisms. *J. Comput. Syst. Sci.*, 78(2):638–650, 2012.

→ *p. 17*

[49] A. A. Bulatov, A. A. Krokhin, and P. Jeavons. The complexity of maximal constraint languages. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC 2001)*, pages 667–674, 2001.

→ *p. 14*

[50] L. Cai, X. Huang, C. Liu, F. A. Rosamond, and Y. Song. Parameterized complexity and biopolymer sequence comparison. *Comput. J.*, 51(3):270–291, 2008.

→ *p. 9*

[51] N. Carmeli and M. Kröll. Enumeration complexity of conjunctive queries with functional dependencies. In B. Kimelfeld and Y. Amsterdamer, editors, *21st International Conference on Database Theory (ICDT 2018)*, volume 98 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

→ *p. 17*

[52] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC 1977)*, pages 77–90. ACM, 1977.

→ *p. 96*

[53] M. Charikar, V. Guruswami, and R. Manokaran. Every permutation CSP of arity 3 is approximation resistant. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC 2009)*, pages 62–73. IEEE Computer Society, 2009.

→ *p. 7*

[54] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998)*, pages 34–43, 1998.

→ *pp. 17, 64, and 65*

[55] C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *J. ACM*, 63(5):40:1–40:65, 2016.

→ *p. 46*

[56] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoret. Comput. Sci.*, 239(2):211–229, 2000.

→ *pp. 17, 93, and 96*

[57] H. Chen and V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 167–181. Springer Berlin / Heidelberg, 2005.

→ *p. 60*

[58] H. Chen and M. Grohe. Constraint satisfaction with succinctly specified relations. *Journal of Computer System Sciences*, 76:847–860, 2010.

→ *p. 14*

[59] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. In *Proceedings of 19th Annual IEEE Conference on Computational Complexity (CCC 2004)*, pages 150–160, 2004.

→ *p. 47*

[60] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 212–221, New York, 2004. ACM.

→ *p. 47*

[61] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

→ *p. 9*

[62] M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Appl. Math.*, 156(3):292–312, 2008.

→ *p. 9*

[63] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A*, 43(1):23–37, 1986.

→ *pp. 61, 65, and 101*

[64] J. Chuzhoy. Excluded grid theorem: Improved and simplified. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, (STOC 2015)*, pages 645–654,

2015.

→ *p. 46*

[65] J. Chuzhoy. Improved bounds for the excluded grid theorem. *CoRR*, abs/1602.02629, 2016.
→ *p. 46*

[66] D. A. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. Comput. Syst. Sci.*, 74(5):721–743, 2008.
→ *p. 60*

[67] B. Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minor and complexity issues. *ITA*, 26:257–286, 1992.
→ *p. 11*

[68] R. Curticapean, H. Dell, and D. Marx. Homomorphisms are a good basis for counting small subgraphs. In H. Hatami, P. McKenzie, and V. King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017), Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017.
→ *pp. 6, 14, and 48*

[69] R. Curticapean and D. Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2014)*, pages 130–139. IEEE Computer Society, 2014.
→ *p. 6*

[70] R. Curticapean and D. Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1650–1669, 2016.
→ *pp. 8, 11, 20, and 21*

[71] R. Curticapean and M. Xia. Parameterizing the permanent: Genus, apices, minors, evaluation mod 2k. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015), Berkeley, CA, USA, 17-20 October, 2015*, pages 994–1009. IEEE Computer Society, 2015.
→ *pp. 14 and 48*

[72] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.
→ *pp. 8, 11, 20, and 21*

[73] M. Cygan, F. V. Fomin, A. Golovnev, A. S. Kulikov, I. Mihajlin, J. Pachocki, and A. Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017.
→ *p. 8*

[74] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
→ *pp. 9, 19, 20, 46, and 94*

[75] M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018.
→ *pp. 11, 20, and 21*

[76] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 150–159, 2011.

→ *pp. 8, 11, 19, and 20*

[77] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.

→ *p. 6*

[78] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 310–326, London, UK, 2002. Springer-Verlag.

→ *p. 104*

[79] V. Dalmau, A. A. Krokhin, and R. Manokaran. Towards a characterization of constant-factor approximable finite-valued CSPs. *J. Comput. Syst. Sci.*, 97:14–27, 2018.

→ *p. 14*

[80] M. de Berg, H. L. Bodlaender, S. Kisfaludi-Bak, D. Marx, and T. C. van der Zanden. A framework for ETH-tight algorithms and lower bounds in geometric intersection graphs. In I. Diakonikolas, D. Kempe, and M. Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2018)*, pages 574–586. ACM, 2018.

→ *p. 8*

[81] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

→ *p. 13*

[82] F. K. H. A. Dehne, M. R. Fellows, F. A. Rosamond, and P. Shaw. Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel $2k$ kernelization for vertex cover. In *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2004.

→ *p. 9*

[83] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

→ *p. 19*

[84] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

→ *p. 19*

[85] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005.

→ *pp. 10 and 46*

[86] I. Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

→ *p. 7*

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

→ *pp. 9, 46, and 94*

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

→ *p. 9*

[89] A. Durand, N. Schweikardt, and L. Segoufin. Enumerating answers to first-order queries over databases of low degree. In R. Hull and M. Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2014)*, pages 121–131. ACM, 2014.

→ *p. 17*

[90] A. Durand and Y. Strozecki. Enumeration complexity of logical query problems with second-order variables. In M. Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 189–202. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

→ *p. 17*

[91] L. Egri, D. Marx, and P. Rzążewski. Finding list homomorphisms from bounded-treewidth graphs to reflexive graphs: a complete complexity characterization. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, pages 27:1–27:15, 2018.

→ *pp. 8, 11, and 20*

[92] E. Eiben, D. Knop, F. Panolan, and O. Suchý. Complexity of the Steiner network problem with respect to the number of terminals. *CoRR*, abs/1802.08189, 2018.

→ *pp. 14 and 48*

[93] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

→ *p. 19*

[94] D. Eppstein and D. Lokshtanov. The parameterized complexity of finding point sets with hereditary properties. *CoRR*, abs/1808.02162, 2018.

→ *pp. 14 and 48*

[95] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

→ *pp. 15, 17, 60, and 93*

[96] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999.

→ *pp. 12 and 14*

[97] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

→ *pp. 49 and 51*

[98] M. R. Fellows, D. Hermelin, F. A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

→ *p. 47*

[99] S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta. Planar graph bipartization in linear time. *Discrete Applied Mathematics*, 156(7), 2008.

→ *p. 19*

[100] W. Fischl, G. Gottlob, and R. Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2018)*, pages 17–32, 2018.

→ *pp. 77 and 80*

[101] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6):716–752, 2002.

→ *p. 69*

[102] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 2006.

→ *pp. 9, 19, 46, 49, 85, 94, and 110*

[103] F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

→ *p. 19*

[104] F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.

→ *p. 8*

[105] F. V. Fomin, P. A. Golovach, and D. M. Thilikos. Approximating acyclicity parameters of sparse hypergraphs. In S. Albers and J.-Y. Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 445–456, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

→ *p. 80*

[106] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.

→ *p. 8*

[107] F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.

→ *p. 20*

[108] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. of AAAI-90*, pages 4–9, Boston, MA, 1990.

→ *pp. 14, 45, and 46*

[109] E. Friedgut and J. Kahn. On the number of copies of a hypergraph in another. *Israel Journal of Mathematics*, 105:251–256, 1998.

→ *pp. 61 and 65*

[110] H. Garcia-Molina, J. Widom, and J. Ullman. *Database System Implementation.* Prentice-Hall, 1999.

→ *p. 64*

[111] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *Comput. J.*, 51(3):372–384, 2008.

→ *p. 9*

[112] M. C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

→ *p. 87*

[113] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree decompositions: Structure, algorithms, and applications. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2005.

→ *p. 60*

[114] G. Gottlob, S. T. Lee, G. Valiant, and P. Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, June 2012.

→ *pp. 17, 59, and 60*

[115] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

→ *pp. 15 and 60*

[116] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64:579–627, 2002.

→ *pp. 15, 17, 60, 77, 78, 79, 93, 96, and 100*

[117] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66:775–808, 2003.

→ *pp. 15, 60, and 78*

[118] G. Gottlob and S. Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008.

→ *pp. 9, 12, and 93*

[119] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25, 1993.

→ *pp. 17 and 64*

[120] J. Gramm, A. Nickelsen, and T. Tantau. Fixed-parameter algorithms in phylogenetics. *Comput. J.*, 51(1):79–101, 2008.

→ *p. 9*

[121] G. Greco and F. Scarcello. The power of tree projections: local consistency, greedy algorithms, and larger islands of tractability. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS 2010)*, pages 327–338, New York, NY, USA, 2010. ACM.

→ *p. 104*

[122] M. Grohe. The structure of tractable constraint satisfaction problems. In R. Kralovic and P. Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium (MFCS 2006)*, volume 4162 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2006.

→ *pp. 12 and 93*

[123] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1, 2007.

→ *pp. 6, 14, 46, 60, 93, 97, 99, and 133*

[124] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 289–298, 2006.

→ *pp. 18, 59, and 77*

[125] M. Grohe and D. Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory Ser. B*, 99(1):218–228, 2009.

→ *pp. 48, 49, and 58*

[126] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

→ *pp. 17, 18, 59, 77, 93, and 96*

[127] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings of the thirty-third annual ACM symposium on Theory of computing (STOC 2001)*, pages 657–666, New York, NY, USA, 2001. ACM Press.

→ *pp. 6, 14, 46, and 60*

[128] J. Guo, S. Hartung, R. Niedermeier, and O. Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013.

→ *pp. 14 and 48*

[129] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In *Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS 2005)*, volume 3608 of *LNCS*, pages 36–48. Springer-Verlag, Aug 2005.

→ *p. 9*

[130] A. Gupta. Improved results for directed multicut. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 454–455, New York, 2003. ACM.

→ *p. 120*

[131] V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.*, 40(3):878–914, 2011.

→ *p. 7*

[132] J. Gustedt, O. A. Mæhle, and J. A. Telle. The treewidth of java programs. In D. M. Mount and C. Stein, editors, *Algorithm Engineering and Experiments, 4th International Workshop (ALENEX 2002)*, volume 2409 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2002.

→ *p. 11*

[133] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996.
→ *p. 17*

[134] M. T. Hajiaghayi and H. Räcke. An $O(\sqrt{n})$-approximation algorithm for directed sparsest cut. *Inform. Process. Lett.*, 97(4):156–160, 2006.
→ *p. 120*

[135] R. Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.
→ *p. 5*

[136] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.
→ *p. 74*

[137] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
→ *p. 7*

[138] T. Hertli. 3-SAT faster and simpler — Unique-SAT bounds for PPSZ hold in general. *SIAM J. Comput.*, 43(2):718–729, 2014.
→ *p. 8*

[139] T. Hertli, R. A. Moser, and D. Scheder. Improving PPSZ for 3-SAT using critical variables. In T. Schwentick and C. Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *LIPIcs*, pages 237–248. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
→ *p. 8*

[140] P. Hliněný. A parametrized algorithm for matroid branch-width. *SIAM J. Comput.*, 35(2):259–277, 2005.
→ *p. 102*

[141] P. Hliněný and S.-i. Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.
→ *p. 102*

[142] P. Hliněný and G. Whittle. Matroid tree-width. *European J. Combin.*, 27(7):1117–1128, 2006.
→ *p. 102*

[143] X. Hu and K. Yi. Towards a worst-case I/O-optimal algorithm for acyclic joins. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2016)*, pages 135–150, 2016.
→ *pp. 17 and 59*

[144] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *Comput. J.*, 51(1):7–25, 2008.
→ *p. 9*

[145] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
→ *pp. 7 and 19*

[146] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

→ *pp. 7, 8, and 132*

[147] Y. E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

→ *p. 17*

[148] K. Iwama, K. Seto, T. Takai, and S. Tamaki. Improved randomized algorithms for 3-SAT. In O. Cheong, K. Chwa, and K. Park, editors, *Algorithms and Computation - 21st International Symposium (ISAAC 2010), Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2010.

→ *p. 8*

[149] S. Iwata. Submodular function minimization. *Math. Program.*, 112(1, Ser. B):45–64, 2008.

→ *p. 118*

[150] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

→ *p. 118*

[151] Y. Iwata and Y. Yoshida. On the equivalence among problems of bounded width. In N. Bansal and I. Finocchi, editors, *23rd Annual European Symposium (ESA 2015)*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015.

→ *pp. 8, 11, and 20*

[152] L. Jaffke and B. M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC 2017)*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017.

→ *pp. 8, 11, 20, and 21*

[153] B. M. P. Jansen, D. Lokshtanov, and S. Saurabh. A near-optimal planarization algorithm. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1802–1811, 2014.

→ *p. 19*

[154] B. M. P. Jansen and D. Marx. Characterizing the easy-to-find subgraphs from the viewpoint of polynomial-time algorithms, kernels, and turing kernels. In P. Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 616–629. SIAM, 2015.

→ *p. 6*

[155] K. Jansen, S. Kratsch, D. Marx, and I. Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013.

→ *pp. 14 and 48*

[156] P. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

→ *p. 14*

[157] M. Joglekar and C. Ré. It's all a matter of degree - using degree information to optimize multiway joins. *Theory Comput. Syst.*, 62(4):810–853, 2018.

→ *pp. 17 and 59*

[158] M. Jones, D. Lokshtanov, M. S. Ramanujan, S. Saurabh, and O. Suchý. Parameterized complexity of directed steiner tree on sparse graphs. *SIAM J. Discrete Math.*, 31(2):1294–1327, 2017.

→ *pp. 14 and 48*

[159] W. Kazana and L. Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013.

→ *p. 17*

[160] M. A. Khamis, H. Q. Ngo, C. Ré, and A. Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Trans. Database Syst.*, 41(4):22:1–22:45, 2016.

→ *pp. 17 and 59*

[161] M. A. Khamis, H. Q. Ngo, and A. Rudra. FAQ: questions asked frequently. In T. Milo and W. Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2016)*, pages 13–28. ACM, 2016.

→ *pp. 17 and 59*

[162] S. Khot. On the power of unique 2-prover 1-round games. In J. H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 767–775. ACM, 2002.

→ *p. 7*

[163] S. Khot and N. K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1. *J. ACM*, 62(1):8:1–8:39, 2015.

→ *p. 7*

[164] J. Kleinberg and E. Tardos. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

→ *p. 19*

[165] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop (WG 2006)*, volume 4271 of *Lecture Notes in Computer Science*, pages 58–67. Springer, 2006.

→ *p. 9*

[166] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

→ *pp. 13 and 93*

[167] P. G. Kolaitis and M. Y. Vardi. A game-theoretic approach to constraint satisfaction. In *AAAI/IAAI*, pages 175–181, 2000.

→ *p. 104*

[168] V. Kolmogorov, A. A. Krokhin, and M. Rolínek. The complexity of general-valued CSPs. *SIAM J. Comput.*, 46(3):1087–1110, 2017.

→ *p. 14*

[169] I. Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium (ICALP 2008)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

→ *p. 9*

[170] K. Kutzkov and D. Scheder. Using CSP to improve deterministic 3-SAT. *CoRR*, abs/1007.1166, 2010.
→ *p. 8*

[171] M. Lampis. Parameterized approximation schemes using graph widths. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014.
→ *p. 19*

[172] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
→ *p. 51*

[173] S. Liu. Chain, generalization of covering code, and deterministic algorithm for k-SAT. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPIcs*, pages 88:1–88:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
→ *p. 8*

[174] D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 777–789, 2011.
→ *pp. 11, 18, and 19*

[175] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.
→ *p. 8*

[176] D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018.
→ *pp. 18 and 19*

[177] D. Lokshtanov, M. S. Ramanujan, S. Saurabh, and M. Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. *CoRR*, abs/1704.04249, 2017.
→ *pp. 14 and 48*

[178] S. Maniu, P. Senellart, and S. Jog. An experimental study of the treewidth of real-world graph data, Sept. 2018. Preprint available at `http://pierre.senellart.com/publications/maniu2019experimental.pdf`.
→ *p. 11*

[179] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Comput. Surv.*, 20(3):191–221, 1988.
→ *p. 17*

[180] D. Marx. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 169–179. IEEE Computer Society, 2007.
→ *pp. 18 and 45*

[181] D. Marx. On the optimality of planar and geometric approximation schemes. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pages 338–348, 2007.
→ *pp. 9 and 132*

[182] D. Marx. Closest substring problems with small distances. *SIAM Journal on Computing*, 38(4):1382–1410, 2008.

→ *pp. 9 and 61*

[183] D. Marx. Approximating fractional hypertree width. In C. Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pages 902–911. SIAM, 2009.

→ *pp. 18 and 77*

[184] D. Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):1–17, 2010.

→ *pp. 17, 18, 77, 80, 93, 96, and 110*

[185] D. Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010.

→ *pp. 18, 45, 99, 127, 132, and 133*

[186] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 735–744, 2010.

→ *pp. 18 and 94*

[187] D. Marx. Tractable structures for constraint satisfaction with truth tables. *Theory of Computing Systems*, 48:444–464, 2011.

→ *pp. 14, 94, 99, 100, 111, 131, and 132*

[188] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

→ *pp. 18 and 94*

[189] D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In E. W. Mayr and N. Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

→ *p. 10*

[190] D. Marx and M. Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In N. Bansal and I. Finocchi, editors, *23rd Annual European Symposium (ESA 2015)*, volume 9294 of *Lecture Notes in Computer Science*, pages 865–877. Springer, 2015.

→ *pp. 14 and 48*

[191] D. Marx and A. Sidiropoulos. The limited blessing of low dimensionality: when $1 - 1/d$ is the best possible exponent for d-dimensional geometric problems. In *30th Annual Symposium on Computational Geometry (SoCG 2014)*, page 67. ACM, 2014.

→ *p. 9*

[192] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory Comput. Syst.*, 43(2):234–253, 2008.

→ *p. 19*

[193] R. A. Moser and D. Scheder. A full derandomization of Schöning's k-SAT algorithm. In L. Fortnow and S. P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of*

Computing (STOC 2011), pages 245–252. ACM, 2011.

→ *p. 8*

[194] J. Nederlof, J. M. M. van Rooij, and T. C. van Dijk. Inclusion/exclusion meets measure and conquer. *Algorithmica*, 69(3):685–740, 2014.

→ *p. 19*

[195] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

→ *pp. 17, 59, 60, and 61*

[196] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013.

→ *pp. 17 and 59*

[197] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

→ *pp. 9, 19, and 94*

[198] S. Oum. Approximating rank-width and clique-width quickly. In *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 49–58, 2005.

→ *pp. 85 and 110*

[199] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

→ *pp. 85 and 110*

[200] S. Oum and P. Seymour. Testing branch-width. *J. Combin. Theory Ser. B*, 97(3):385–393, 2007.

→ *pp. 85, 102, and 110*

[201] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

→ *p. 74*

[202] M. Patrascu and R. Williams. On the possibility of faster SAT algorithms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1065–1075, 2010.

→ *pp. 8 and 132*

[203] M. Pilipczuk and M. Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *TOCT*, 10(3):13:1–13:18, 2018.

→ *pp. 14 and 48*

[204] V. Poosala and Y. E. Ioannidis. Estimation of query-result distribution and its application in parallel-join load balancing. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 448–459. Morgan Kaufmann, 1996.

→ *p. 17*

[205] B. Reed. Tree width and tangles: A new connectivity measure and some applications. In R. Bailey, editor, *Surveys in Combinatorics*, volume 241 of *LMS Lecture Note Series*, pages

87–162. Cambridge University Press, 1997.

→ *pp. 49 and 82*

[206] J. Rhadakrishnan. Entropy and counting. IIT Kharagpur, Golden Jubilee Volume, on Computational Mathematics, Modelling and Algorithms (Ed. JC Mishra), Narosa Publishers, New Delhi, 2001. Available online at http://www.tcs.tifr.res.in/∼jaikumar/Papers/EntropyAndCounting.pdf.

→ *p. 61*

[207] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

→ *p. 5*

[208] N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Combin. Theory Ser. B*, 41(1):92–114, 1986.

→ *p. 46*

[209] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994.

→ *p. 46*

[210] L. Roditty and V. V. Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th Annual ACM on Symposium on Theory of Computing (STOC 2013)*, pages 515–524, 2013.

→ *p. 8*

[211] F. Scarcello, G. Gottlob, and G. Greco. Uniform constraint satisfaction problems and database theory. In *Complexity of Constraints*, pages 156–195, 2008.

→ *pp. 12 and 93*

[212] T. J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. ACM, New York, 1978.

→ *p. 14*

[213] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B*, 80(2):346–355, 2000.

→ *p. 118*

[214] A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency.*, volume 24 of *Algorithms and Combinatorics*. Springer, Berlin, 2003.

→ *p. 63*

[215] N. Schweikardt, L. Segoufin, and A. Vigny. Enumeration for FO queries over nowhere dense graphs. In J. V. den Bussche and M. Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (2018)*, pages 151–163. ACM, 2018.

→ *p. 17*

[216] A. D. Scott and G. B. Sorkin. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optimization*, 4(3-4):260–287, 2007.

→ *p. 19*

[217] P. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.

→ *pp. 77, 78, and 82*

[218] C. Sloper and J. A. Telle. Towards a taxonomy of techniques for designing parameterized algorithms. In *Parameterized and Exact Computation, Second International Workshop (IWPEC 2006)*, volume 4169 of *Lecture Notes in Computer Science*, pages 251–263. Springer, 2006.

→ *p. 9*

[219] C. Sloper and J. A. Telle. An overview of techniques for designing parameterized algorithms. *Comput. J.*, 51(1):122–136, 2008.

→ *p. 9*

[220] L. Sunil Chandran and F. Grandoni. Refined memorization for vertex cover. *Inform. Process. Lett.*, 93(3):125–131, 2005.

→ *p. 9*

[221] A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theor. Comput. Sci.*, 137(2):253–268, 1995.

→ *p. 21*

[222] R. E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55(2):221–232, 1985.

→ *pp. 85 and 87*

[223] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings of the 3rd Workshopon Algorithms and Data Structures (WADS 1993)*, volume 709 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 1993.

→ *p. 20*

[224] J. Thapper and S. Zivny. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37:1–37:33, 2016.

→ *p. 14*

[225] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24, 2005.

→ *p. 19*

[226] M. Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998.

→ *p. 11*

[227] J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009)*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

→ *pp. 11, 19, and 20*

[228] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

→ *pp. 72 and 90*

[229] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory ((ICDT) 2014)*, pages 96–106, 2014.
→ *pp. 17, 59, 60, and 61*

[230] S. H. Whitesides. An algorithm for finding clique cut-sets. *Inform. Process. Lett.*, 12(1):31–32, 1981.
→ *p. 87*

[231] R. Williams. Finding paths of length k in $O(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.
→ *p. 9*

[232] V. V. Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *LIPIcs*, pages 17–29, 2015.
→ *p. 8*

[233] M. Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, (VLDB 1981)*, pages 82–94. IEEE Computer Society, 1981.
→ *pp. 15, 17, 60, and 93*

[234] D. Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017), Berkeley, CA, USA, October 15-17, 2017*, pages 331–342, 2017.
→ *p. 14*