Structurally Parameterized Tight Bounds and Approximation for Generalizations of Independence and Domination

Ioannis Katsikarelis

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'Université Paris-Dauphine

Structurally Parameterized Tight Bounds and Approximation for Generalizations of Independence and Domination

Soutenue par
Ioannis KATSIKARELIS
Le **12/12/2019**

École doctorale n°543
ED de Dauphine

Spécialité
Informatique

Composition du jury :

Mathieu LIEDLOFF Maître de Conférences, Université d'Orléans	*Rapporteur*
Ignasi SAU VALLS Chargé de Recherche CNRS, LIRMM Montpellier	*Rapporteur*
Bruno ESCOFFIER Professeur, Sorbonne Université	*Examinateur*
Ioan TODINCA Professeur, Université d'Orleans	*Président du jury*
Michail LAMPIS Maître de Conférences, Université Paris-Dauphine	*Codirecteur de thèse*
Vangelis Th. PASCHOS Professeur, Université Paris-Dauphine	*Directeur de thèse*

Dauphine | PSL

Acknowledgements

As an expression of my sincere gratitude towards (1) my **supervisors** Vangelis Th. Paschos and Michail Lampis; (2) the members of my **jury** Bruno Escoffier, Mathieu Liedloff, Ignasi Sau Valls and Ioan Todinca; (3) my **co-authors** (and Sakura people) Rémy Belmonte, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Eun Jung Kim, Valia Mitsou, Hirotaka Ono, Yota Otachi and Florian Sikora; (4) all of **LAMSADE** and anyone else who may have contributed to the making of the present thesis: *merci beaucoup !*

Contents

1 Introduction

The aim of computational complexity theory is the categorization of mathematical problems into classes according to the worst-case running-times of the algorithms that solve them. In the classical setting problems are considered *tractable*, that is, polynomial-time solvable, if there exists an algorithm whose running-time can be expressed as a polynomial function on the size n of the input. On the other hand, the *intractable* problems (commonly NP-hard[1]) are those for which a polynomial-time algorithm is considered unlikely, based on the (widely believed) conjecture that P≠NP: if the 3-SAT problem does not admit a deterministic polynomial-time algorithm, then a *reduction* from 3-SAT to another problem, i.e. a transformation from one problem's instances to the other's demonstrating their equivalence in terms of computational complexity, would imply that the latter problem does not admit a polynomial-time algorithm as well.

Taking the above considerations one step further leads to the formulation of another (also widely believed) conjecture, the *Exponential Time Hypothesis* (ETH): it conjectures there is no *subexponential* algorithm for 3-SAT, i.e. no algorithm of running-time $2^{o(n)} \cdot n^{O(1)}$. If this hypothesis is true, then P is not equal to NP and any algorithm for 3-SAT will require at least exponential time, in the worst case. A slightly more demanding (and not-so-widely believed) version, the *Strong Exponential Time Hypothesis* (SETH) has also been formulated and asserts that (general) SAT does not admit an algorithm of running-time $(2-\epsilon)^n \cdot n^{O(1)}$ for any constant $\epsilon > 0$.

As with the assumption that P≠NP, the main importance of the ETH and SETH, however, lies not in whether these may actually be true or not: in a similar manner as in the classification of problems into polynomial-time solvable or NP-hard, we can use the ETH and SETH as starting points in showing, via hardness reductions, the non-existence of any algorithm of some *specific* running-time below a certain threshold and in this way derive results that preclude the existence of such algorithms for a given problem (a *lower bound*). Combining results of this type with algorithms whose worst-case (or *upper bounded*) running-times exactly match these lower bounds, we can precisely identify the complexity of a given problem and justify the *optimality* of our approach. Naturally, any such statements we make will be subject to the above assumptions, meaning our results will imply the proposed algorithms are optimal, unless significant progress is made in our understanding of the fundamental principles of computation.

[1] So as not to overload this preface with notation, all formal definitions of the technical terms we freely discuss here are postponed until Section 2.1, along with the related bibliographical references.

In this way, we can further classify problems according to their *exact* computational requirements, especially in the case of the SETH that offers a more precise source at the cost of a more ambitious (and therefore more likely to be incorrect) assumption. This refinement importantly allows us to advance our understanding of the available options for tackling a provably demanding computational problem and can potentially lead to practical improvements in applied settings, yet it is the possibility of precisely characterizing the underlying complexity features of mathematical problems that will be mostly of interest to us here. Having shown both an upper and a lower bound of matching complexity functions for a given problem (i.e. bounds that are *tight*), we can usually identify a uniformity in the mathematical structure of the two proofs that is not arbitrary, as in the optimality of a reduction that will produce instances (almost) explicitly constructed to hinder the efforts of the algorithm whose running-time exactly matches the reduction's lower bound (and vice-versa). Results such as these can be seen as implying that an aspect of the problem's essence has been undeniably identified, since their validity does not depend on the particular methods employed by their designer.

Parameterization and Approximation Being able to characterize the intractability of a problem according to a particular mode of computation does not mean we have exhausted all possibilities for addressing it. Regarding a problem as intractable if the required running-time of any algorithm for its exact solution is at least exponential in the size of the input can thus lead to other directions for advancing our comprehension of the intricate mechanisms that regulate complex combinatorial problems: *parameterization* and *approximation*. On one hand and in search of a finer characterization of the necessary amount of computation needed for the exact and optimal resolution of a computational problem, we could allow the complexity functions to grow indeed exponentially, but not on the size n of the input (that must naturally be considered too large and impractical). Via parameterization we study the complexity of problems in terms of other parameters that specify their properties than simply the size of the input, parameters whose (bounded) size would not preclude practical computations of an exponential number. On the other hand, relaxing the requirement that the solutions returned by our algorithms are necessarily the best possible (being of measurable quality for *optimization* problems) and focusing on keeping the running-times polynomial on the size of the input, we enter the realm of approximation. Here, our solutions must be accompanied by mathematical guarantees of remaining above certain quality thresholds (a worst-case *approximation ratio*).

In parameterized complexity, similarly to the classification of problems as NP-hard or polynomial-time solvable, a mathematical problem that is solved by an algorithm whose complexity can be expressed as a function of the form $f(k) \cdot n^{O(1)}$, where k is the chosen parameter and f is any computable function, belongs to the class of *fixed-parameter tractable* problems (FPT). Depending on the problem, the functions f may take on many forms, being exponential in the majority of cases. This implies that should the size of the parameter under consideration not be too large, for a given instance of a problem to be solved, then an FPT algorithm that solves it could be considered usable (perhaps even practical), while also providing important refinements on the complexity landscape in general. Common parameters include the size of an optimal solution (the *standard* parameterization), as well as a variety of *structural* measures that characterize the inherent structure of the input instance. Here, a problem is considered intractable if it can be shown (via *parameterized reductions*, also maintaining a close relationship between parameters)

to be as hard to solve as any problem that is complete for a level of the *W-hierarchy* of complexity classes (considered an analogue of NP-hardness), i.e. if it is unlikely to be FPT.

The theory of approximation algorithms concerns itself with the complementary side of intractability: identifying the best possible worst-case bounds on the quality of a returned solution that can be obtained if the running-time is confined to the polynomials in n. These bounds are commonly expressed as the ratio between the worst-case quality of a returned solution and that of an optimal solution for the same instance. On the 'hardness' side, it is possible to show (via *approximation-preserving reductions*) that there is no polynomial-time algorithm achieving a certain ratio for a given problem, under standard complexity assumptions, thus establishing its *inapproximability*. Common ratios in results of this type include inapproximability to specific constants, to any possible constant ratio, as well as a ratio that is a function of n. The allowed running-time for an approximation algorithm is commonly polynomial in n, yet it is possible to allow other functions (such as FPT running-times) in order to propose alternatives to exact computation, should a problem remain intractable beyond the polynomial-time boundary.

Returning once more to the ETH and SETH, we may observe that in conjunction with the refined complexity analysis performed by the studies of parameterization it is possible to obtain improved lower bounds of increased precision on the required running-time of any algorithm for a given problem. Both hypotheses can be considered as assumptions on the complexity of q-SAT parameterized by the number of variables n and a parameterized reduction to another problem in this case (i.e. where the size of the parameter is bounded by an appropriate function of n) would yield results on the non-existence of a subexponential (in the size of the parameter) algorithm for the problem in question. Thus parameterized problems can be further categorized in terms of the exact functions that determine their complexity with respect to the variety of possible parameters that partake in their intractability, leading to a much-improved understanding of the field of computational complexity.

Covering and Packing problems: In this thesis we focus on the well-known graph-theoretical problems (k, r)-Center and d-Scattered Set that generalize the concepts of vertex *domination* and *independence* over larger distances within the graph. In the Dominating Set problem we are looking for the *smallest* subset of vertices such that every *other* vertex is connected to at least one vertex in the subset. On the other hand, in Independent Set we require the *largest* subset such that no pair of vertices *in the subset* have an edge between them. Intuitively, a dominating set must *cover* the rest of the graph based on the combined adjacency of its vertices to those of the complement, while in an independent set we must be able to *pack* as many pairwise non-adjacent vertices as possible. Both problems exhibit firm intractability: they are NP-hard, their standard parameterizations W-hard and generally inapproximable in polynomial- as well as FPT-time. On the positive side, both problems turn out to be FPT when parameterized by the most widely-used structural parameters. This means that when the input graph is of restricted structure, both problems can be efficiently, as well as exactly solved.

The generalizations of these well-studied notions that we will be examining here are based on extending the central *distance parameter* in their definitions to unbounded values. In (k, r)-Center we are asked for the smallest set that covers the graph *at distance* r and in d-Scattered Set we must pack as many vertices as possible *at distance* d from each other. This means our perspective here must expand to consider the influence of

vertices taking part in the solution over larger areas within the graph, as the significance of adjacency lies now with *paths* instead of edges. As a preliminary remark, it turns out that reachability between vertices is too responsive a property to small shifts in their exact location, making the existence of collections of paths of non-trivial length crucially depend on the exact shape of *local* structures and therefore the behaviour of both problems will diverge (significantly) from their base cases when r, d are large. This effect is reflected in our results, since both problems become intractable even for graphs of significantly restricted structure, if the value of the distance parameter is not bounded in each case. Thus the above-mentioned algorithms are efficient only for small, fixed values (e.g. $r = 1, d = 2$), motivating our subsequent analysis.

Our Scope: We will consider the problems (k, r)-Center and d-Scattered Set, paying particular attention to how their complexity is affected by the distance parameters and to the available options for their exact and/or efficient computation. Since our problems are in fact generalizations of Dominating Set and Independent Set, our results can be seen to match (and sometimes even improve) the state-of-the-art for these problems.

In the first part of the thesis we maintain a parameterized viewpoint: we examine the standard parameterization, as well as the most commonly used graph measures treewidth tw, clique-width cw, tree-depth td, vertex cover vc and feedback vertex set fvs. We offer hardness results that show there is no algorithm of running-time below certain bounds (subject to the ETH, SETH), produce essentially optimal algorithms of complexity that matches these lower bounds and further attempt to offer an alternative to exact computation in significantly reduced running-time by way of approximation. In particular, for (k, r)-Center we show the following:

- A dynamic programming algorithm of running-time $O^*((3r + 1)^{\mathrm{cw}})$, assuming a clique-width expression of width cw is provided along with the input, and a matching SETH-based lower bound that closes a complexity gap for Dominating Set parameterized by cw (for $r = 1$).
- W[1]-hardness and ETH-based lower bounds of $n^{o(\mathrm{vc}+k)}$ for edge-weighted graphs and $n^{o(\mathrm{fvs}+k)}$ for unweighted graphs. This shows the importance of bounding the value of r. Also for the unweighted case, we give an $O^*(5^{\mathrm{vc}})$-time FPT algorithm based on solving appropriate Set Cover sub-instances.
- A tight ETH-based lower bound of $O^*(2^{O(\mathrm{td})^2})$ for parameterization by td.
- Algorithms computing for any $\epsilon > 0$, a $(k, (1+\epsilon)r)$-center in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, or $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$, if a (k, r)-center exists in the graph, assuming a tree decomposition of width tw is provided along with the input.

Then for d-Scattered Set and applying similar methods we show:

- A dynamic programming algorithm of running-time $O^*(d^{\mathrm{tw}})$ and a matching lower bound based on the SETH, that generalize known results for Independent Set.
- W[1]-hardness for parameterization by $\mathrm{vc} + k$ for edge-weighted graphs, as well as by $\mathrm{fvs} + k$ for unweighted graphs, again showing the importance of bounding d. These are complemented by FPT-time algorithms for the unweighted case that use ideas related to Set Packing, of complexity $O^*(3^{\mathrm{vc}})$ for even d and $O^*(4^{\mathrm{vc}})$ for odd d.

- A tight ETH-based lower bound of $O^*(2^{O(\mathrm{td})^2})$ for parameterization by td, as above.
- An algorithm computing, for any $\epsilon > 0$, a $d/(1+\epsilon)$-scattered set in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, if a d-scattered set exists in the graph, assuming a tree decomposition of width tw is provided in the input.

We note these results are comparable to those for (k, r)-Center, since our work on d-Scattered Set can be considered as a continuation of the above. As we will see, both problems are similarly affected by distance-based generalizations and are thus responsive to similar techniques.

In the second part of the thesis we focus on d-Scattered Set and in particular its (super-)polynomial (in-)approximability: we are interested in the exact relationship between an achievable approximation ratio ρ, the distance parameter d, and the running-time of any ρ-approximation algorithm expressed as a function of the above and the size of the input n. Following this, we consider strictly polynomial running-times and graphs of bounded maximum degree as well as bipartite graphs. Specifically we show:

- An exact exponential-time algorithm of complexity $O^*((ed)^{\frac{2n}{d}})$, based on an upper bound on the size of any solution.
- A lower bound on the complexity of any ρ-approximation algorithm of (roughly) $2^{\frac{n^{1-\epsilon}}{\rho d}}$ for even d and $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ for odd d, under the randomized ETH.
- ρ-approximation algorithms of running-times $O^*((e\rho d)^{\frac{2n}{\rho d}})$ for even d and $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ for odd d that (almost) match the above lower bounds.
- A lower bound of $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ on the approximation ratio of any polynomial-time algorithm for graphs of maximum degree Δ, being the first lower bound of this type, as well as an improved upper bound of $O(\Delta^{\lfloor d/2 \rfloor})$.
- A polynomial-time $2\sqrt{n}$-approximation for bipartite graphs and even values of d, that complements known results by considering the only remaining open case.

The rest of the thesis is organized in the following way: the theoretical background, definitions and preliminary results we require are given in Section 2.1, while we discuss related work in Section 2.2; Chapter 3 deals with the (k, r)-Center problem, presenting results published in [67] (originally in [63]) that can also be found in [62]; parameterized results on d-Scattered Set that were originally published in [68] and can also be found in [64] are presented in Chapter 4; (in)approximability results on the same problem are presented in Chapter 5 and can also be found in [65, 66]; a summary of our results and some discussion on open problems may be found in Chapter 6.

2 Preliminaries

2.1 Definitions

Computational Complexity background

A *decision problem* is a *language* commonly defined on the binary alphabet, i.e. a subset of $\{0,1\}^*$. Given some appropriate encoding (a *string* of binary characters) that fully describes an *instance*, that is a specific yet abstracted situation, and on which a formal question can be asked with possible answers only of the type "yes" or "no", the language of a decision problem consists of all such binary strings that encode instances for which the answer to the question is positive and no strings encoding instances for which the answer is negative. A particular instance of a problem is a *word* on the same alphabet and the word is part of the language only if the word's answer to the problem's question is "yes".

A *Boolean expression* ϕ is a mathematical formula built on n binary *variables* $x_i \in \{0,1\}, i \in [1,n]$, parentheses and logical operators: *conjunction* denoted by $\wedge$ ("and"), *disjunction* denoted by $\vee$ ("or"), and *negation* denoted by $\neg$ ("not"). A conjunction of two variables evaluates to 1 if *both* variables are set to 1, a disjunction of two variables evaluates to 1 if *at least one* variable is set to 1 and a negation inverts the value of the variable it precedes. An *assignment* is a string of length n of values for the variables of ϕ, or a member of $\{0,1\}^n$. A formula is said to be *satisfiable* if there is an assignment to the variables such that the expression evaluates to 1, i.e. a *satisfying assignment*. A *literal* is an appearance of a variable x_i in the formula, along with its negation where present. A formula is in *Conjunctive Normal Form* (CNF) if it is a conjunction of (usually m) disjunctions of literals, that we refer to as the *clauses*.

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_3 \vee \neg x_2 \vee x_6) \tag{2.1}$$

Formula ϕ above is in CNF and has $n = 6$ variables and $m = 3$ clauses. A satisfying assignment for ϕ is 001000, as it makes every clause evaluate to 1.

Definition 1. *In the* Satisfiability *problem (SAT), we are given a Boolean expression ϕ on n binary variables and m clauses in CNF form and are asked if ϕ is satisfiable.*

We let *q-SAT* refer to the version of SAT where each clause of ϕ is of size at most q, i.e. ϕ is a conjunction of m disjunctions, each of at most q literals and still on a total number of n variables. Formula ϕ above is an instance of 3-SAT, since the maximum number of

literals in any clause is 3. For a given problem Π and an instance $x \in \Pi$, a *solution* (or *certificate*) is a string y that encodes the part of the instance that justifies the correct answer to the problem's question as positive. For an instance ϕ of SAT, a solution y is a string of length n that describes a satisfying assignment.

An *algorithm* is an unambiguous procedure for identifying a solution (its *output*) to any given instance of a problem (its *input*). The number of operations applied or calculations performed by an algorithm is called its *running-time* and is commonly indirectly expressed as belonging to a set of functions of some appropriate measure of the input's size, based on the functions' *order* and employing the asymptotic notation: for two positive functions f, g, it is $f(n) \in O(g(n))$ if there exist positive constants c, n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$, while $f(n) \in o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. In words, the notation $f(n) \in O(g(n))$ (we also write $f(n) = O(g(n))$) means that $f(n)$ is *upper-bounded* by $g(n)$ (up to constant multiplicative factors) and the notation $f(n) \in o(g(n))$ (also $f(n) = o(g(n))$) means that the rate of growth of $f(n)$ is insignificant compared to that of $g(n)$.

We say that an algorithm for instances of a problem of size n (for an appropriate definition of *size* in each case) is a *polynomial-time* (resp. *exponential-time*) algorithm if its running-time expressed as a function of n belongs to $O(g(n))$ for any polynomial (resp. exponential) on n function $g(n)$. We refer to the set of functions describing the order of the running-time function of an algorithm as the algorithm's *complexity*, while a problem's *computational complexity* is the *best*, i.e. the lowest (inclusion-wise) complexity of an algorithm that solves the problem's *worst-case* instances, i.e. those instances of the problem that will require the maximum number of calculations/operations in order to be decisively solved.

A *reduction* is an algorithm that transforms an instance x of a problem Π_1 to an "equivalent" instance y of problem Π_2, with appropriate significations of equivalence giving rise to different types of reductions, each suited to its particular purpose and the types of problems its function is to relate. Specifically, for two languages Π_1, Π_2 encoding decision problems and defined over alphabets Σ_1, Σ_2, a *many-one reduction* from Π_1 to Π_2 is a total computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that for a word $x \in \Sigma_1^*$, it is $x \in \Pi_1$ if and only if $y \in \Pi_2$ for $y = f(x) \in \Sigma_2^*$. This means the reduction's function must compute instances y of the problem Π_2 for which the answer to the question of problem Π_2 is "yes", if and only if the answer to the question of problem Π_1 for the reduction's input instance x is also "yes".

The existence of such an algorithm means that to solve problem Π_1, one may use an algorithm for problem Π_2 on the output of the reduction. Introducing notions of *efficiency*, i.e. limits on allowed running-time functions, we can infer for a problem Π_1 *reducible* to a problem Π_2, that Π_1 is as computationally involved, or as efficiently solvable as Π_2: if there exists a polynomial-time algorithm for problem Π_2 and a polynomial-time many-one reduction from Π_1 to Π_2, then the algorithm for Π_1 that applies the former on the output of the latter is also a polynomial-time algorithm. Thus particular types of reductions can be seen to form an *equivalence relation* (a reflexive and transitive binary relation, a *preorder*) on a set of problems, whose *equivalence classes* (sets of equivalence based on this relation) can be used to define *complexity classes*, i.e. categorizations of problems according to their computational complexity as defined above. From this moreover emerge the computational notions of *hardness* and *completeness*: a problem Π is called Γ-*hard* for a complexity class Γ, under a specific type of reduction, if there exists a reduction of this type from *any problem* in Γ to Π. If a problem is shown to be Γ-hard and also a member

of class Γ, then it is called Γ-*complete* for this type of reduction. Complete problems can thus be naturally seen as representatives of their class.

The most common differentiation in terms of worst-case algorithmic running-time between problems is focused on their identification as polynomial or exponential. This leads to the definitions of the corresponding complexity classes: these problems whose worst-case instances admit an exponential-time algorithm (i.e. $O(2^{p(n)})$ for any polynomial $p(n)$) belong to the class *EXPTIME*, while the subset of these with a worst-case complexity that is actually $O(p(n))$ belongs to *P*. Our interest in differentiating between these types of running-times is not arbitrary: an algorithm of exponential (in the size of the input) complexity is allowed to perform calculations on *any possible* arrangement of the input's particular objects of interest and is thus guaranteed to solve most of the interesting combinatorial problems, while in strictly polynomial running-time an algorithm must be able to find a solution after only a specific number of *iterations* over the set of input objects. An exponential-time algorithm for instances ϕ of SAT on n variables can simply calculate the truth value of ϕ for each of the 2^n possible assignments to the variables and decide whether a satisfying assignment exists if at least one of these makes ϕ true. Thus we know that SAT belongs to EXPTIME, but since no polynomial-time algorithm has (yet) been discovered for SAT, it is unknown whether SAT also belongs to P.

The class of decision problems whose certificates (of length at most polynomial on the size of the input) can be *verified* in polynomial time as in fact encoding the part of the input instance that justifies a correct answer to the problem's question as "yes" is called *NP*. All problems in P are also in NP. For SAT, an assignment to the variables is such a certificate of length n and the truth value of ϕ can be verified in polynomial time, meaning SAT is in NP. The famous Cook-Levin theorem states that SAT is also *NP-hard*, making it the first *NP-complete* problem. It is generally conjectured that no NP-complete problem is in P, meaning that there is no polynomial-time algorithm for any of these problems, as the existence of a polynomial-time algorithm for one of them would also imply the existence of such an algorithm for all other problems that are reducible to it in polynomial time. This central complexity assumption acts as a starting point for much subsequent theory.

In a more quantitative manner, the *Exponential Time Hypothesis* (ETH) implies that 3-SAT cannot be solved in (*subexponential*) time $2^{o(n)}$ on instances with n variables. The *Strong Exponential Time Hypothesis* (SETH) implies that for all $\epsilon > 0$, there exists an integer q such that q-SAT cannot be solved in time $(2-\epsilon)^n$ on instances with n variables. More formally, for each $q \geq 2$, let s_q be the infimum of the real numbers γ for which q-SAT can be solved in time $O(2^{\gamma n})$, on instances of size n. Then it is $s_2 = 0$ (as 2-SAT is solvable in polynomial time) and the numbers $s_3 \leq s_4 \leq \ldots$ form a monotonic sequence that is bounded above by 1, meaning they must converge to a limit s_∞. The ETH is the conjecture that $s_q > 0$ for every $q > 2$, or, equivalently, that $s_3 > 0$. The SETH is the conjecture that $s_\infty = 1$.

A *randomized* (or *probabilistic*) algorithm employs a probability distribution in its decision-making process and is thus not *deterministic*. Usually this involves additional input in the form of uniformly random bits and the *probabilistic error* is found in the correctness of the answers given and the validity of produced solutions. The class of decision problems solvable by a probabilistic algorithm in polynomial time with an error probability bounded away from 1/3 for all instances is called *BPP*. This means for problems in BPP that the algorithms solving them are guaranteed to run in polynomial time, can make random decisions and each of their applications has a probability $\leq 1/3$ of giving the

wrong answer, whether the answer is "yes" or "no". Note that any number in $[0, 1/2)$ gives rise to the same class since probabilistic algorithms can be applied repeatedly. All problems in P are obviously also in BPP. The relationship between BPP and NP is unknown, yet it is also widely conjectured that NP$\not\subset$BPP. Moreover, a similar conjecture to the ETH can be proposed when considering randomized algorithms.

Some of the most interesting (and applicable) decision problems pose questions related to whether the maximum or minimum attainable value of a given function for a given instance is above or below a certain threshold k. Related to each such decision problem is its corresponding *optimization problem*. The MaxSAT problem asks for an assignment to the variables of instance ϕ that satisfies the maximum number of clauses, while the related *decision version* asks if that maximum is $\geq k$. Instances of such problems can have any number of *feasible solutions*, each associated with a particular *value* of a computable *objective function* defined on it. For MaxSAT, each assignment to the variables of ϕ is associated with the number of clauses it satisfies. An *optimal* solution is one for which the problem's objective function attains its extremal value, being the minimum for *minimization* and the maximum for *maximization* problems. An optimal assignment for MaxSAT satisfies the maximum possible number of clauses for the given instance ϕ. The decision version of MaxSAT is also called NP-hard because an algorithm that solves it can be used to solve SAT, which is NP-complete.

A *ρ-approximation* algorithm for an optimization problem computes a solution whose value is guaranteed to be at most a multiplicative factor ρ (the algorithm's *approximation ratio*) away from the value of an optimal solution for any given instance of the problem. Here we consider ratios $\rho > 1$ for both minimization and maximization problems and thus the best achievable ratios are always as close to 1 as possible. Formally, for an instance I of an optimization problem Π, let $ALG(I)$ be the value of the problem's objective function on a solution obtained by a ρ-approximation algorithm and $OPT(I)$ be that of an optimal solution for I. Then it is $\rho \geq \frac{OPT(I)}{ALG(I)}$ for any instance I of a maximization problem Π, while in the case of minimization problems it is $\frac{1}{\rho} \leq \frac{OPT(I)}{ALG(I)}$.

A *Polynomial-Time Approximation Scheme* (PTAS) is an algorithm that produces a solution whose value is within a factor of $(1 + \epsilon)$ from the optimal for a given $\epsilon > 0$ and any instance of an optimization problem in polynomial time. The running-time of a PTAS must be polynomial in the size of the input n for every fixed ϵ. This includes algorithms of running-time $O(n^{1/\epsilon})$. The class PTAS contains all problems that admit a polynomial-time approximation scheme and is a subset of *APX*, the class of problems that are approximable to some constant factor. Unless P=NP, there exist problems that are in APX but without a PTAS, so PTAS$\subsetneq$APX. The MaxSAT problem is APX-complete and therefore admits no PTAS under the same assumption.

An *approximation-preserving reduction* from an optimization problem $\Pi_1 \subseteq \Sigma_1^*$ to another optimization problem $\Pi_2 \subseteq \Sigma_2^*$ is a pair of functions $f : \Sigma_1^* \to \Sigma_2, g : \Sigma_2^* \to \Sigma_1^*$, where f maps an instance x of Π_1 to an instance y of Π_2 and g maps a solution for y to a solution for x. Such algorithms must preserve more than the validity of solutions when reducing from one problem to another, as some guarantee on the worst-case relationship between the value of any solution to that of an optimal solution must also be maintained. What is of interest in this type of transformation between problem instances, especially when considering approximation algorithms with ratios that are functions of the input size n, is the way in which the objective value of valid/optimal solutions relates to the changing size of the instance. The approximation-preserving reductions we present here

will maintain an equivalence between the objective values of solutions for the input and produced instances, with a marked increase in the size of the latter compared to that of the former.

For a minimization problem $\Pi \subseteq \Sigma^*$, a *gap-introducing reduction* from SAT to Π with *gap* functions f, α is a polynomial-time algorithm that transforms an instance ϕ of SAT to an instance $x \in \Pi$, such that:

- if ϕ is satisfiable, $OPT(x) \leq f(x)$, and
- if ϕ is not satisfiable, $OPT(x) > \alpha(|x|) \cdot f(x)$.

Accordingly, for a maximization problem Π, it must be:

- if ϕ is satisfiable, $OPT(x) \geq f(x)$, and
- if ϕ is not satisfiable, $OPT(x) < \frac{f(x)}{\alpha(|x|)}$.

On the hardness side, the *gap* $\alpha(|x|)$ is the dual notion of the approximation ratio for the algorithmic side: a gap-introducing reduction from SAT shows that it is NP-hard to approximate the problem within a *hardness factor* of $\alpha(|x|)$, or, equivalently, that the problem admits no polynomial-time $\alpha(|x|)$-approximation. Note that we maintain $\alpha(|x|) \geq 1$ in both cases as we also consider approximation ratios $\rho > 1$.

A *parameterized problem* is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$ on finite alphabet Σ defined along with a number characterizing some aspect of the instance and referred to as the *parameter*. An instance (ϕ, k) of the decision version of MaxSAT parameterized by the maximum number of simultaneously satisfiable clauses, i.e. the size k of an optimal solution for ϕ (generally called the *standard* parameterization), is a word of the parameterized language if there is an assignment that satisfies k clauses of ϕ.

A parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is *Fixed-Parameter Tractable* (FPT) if there exist a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c and a *parameterized algorithm* that for any instance $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether $(x, k) \in \Pi$ in time bounded by $f(k) \cdot O((|x| + k)^c)$, that is polynomial in the size of the input $(|x| + k)$. The running-time functions f are commonly exponential in the size of the parameter. The complexity class FPT contains all parameterized problems admitting algorithms with running-times of this form. When referring to FPT running-times we mostly use the $O^*(\cdot)$-notation to imply omission of factors polynomial in the size of the input n and focus on the part of the running-time expressed by the function f.

For two parameterized problems $\Pi_1, \Pi_2 \subseteq \Sigma^* \times \mathbb{N}$, a *parameterized reduction* from Π_1 to Π_2 is an algorithm that runs in time bounded by $f(k) \cdot |x_1|^{O(1)}$ and produces an instance $(x_2, k_2) \in \Pi_2$ from an instance $(x_1, k_1) \in \Pi_1$, such that $k_2 \leq g(k_1)$, for some computable function g. A parameterized problem reducible by such a reduction to a problem in the class FPT is also a member of the class. Moreover, if in the above definition of FPT the constant c is replaced by a computable function $g : \mathbb{N} \to \mathbb{N}$, we arrive at the definition of *slice-wise polynomial* (XP) problems and the corresponding complexity class. This means there exists an algorithm that solves each problem in XP in time that is bounded by $f(k) \cdot O((|x| + k)^{g(k)})$. It is (provably) FPT$\subsetneq$XP.

In parameterized complexity, the class FPT plays a role analogous to that of the class P in classical complexity theory, with XP considered analogous to EXPTIME. Once more, the distinction between fixed-parameter tractable and slice-wise polynomial functions is

not arbitrary: consider as an example the case of standard parameterization by solution size k, where an XP-time algorithm can compute all the n^k candidate subsets and decide if a solution of this size exists. The analogous role to that of the class NP is played here by the *W-hierarchy*: each class W[t] in the hierarchy is defined as the class of parameterized problems reducible to a version of SAT defined on boolean circuits of *weft* t. We omit the definitions of notions related to circuits and provide the definitions of the following parameterized problems instead, each one complete for a level t of the hierarchy.

For a formula ϕ, the *weight* of a particular assignment is the number of variables to which the assignment gives a value of 1. The Weighted 2-Satisfiability problem is the version of 2-SAT parameterized by k, where we are asked to find a satisfying assignment of weight at most k for an input formula where each clause is of size at most 2. Weighted 2-Satisfiability is W[1]-complete. A Boolean formula ϕ is *t-normalized* if it is a conjunction of disjunctions of conjunctions and so on, alternating for t levels. Formally, letting A_0, B_0 be the the set of formulas consisting of a single literal, A_t is defined for $t \geq 1$ as the set of formulas consisting of the disjunction of any number of B_{t-1} formulas and B_t is defined for $t \geq 1$ as the set of formulas consisting of the conjunction of any number of A_{t-1} formulas. B_t is then exactly the set of t-normalized formulas. The Weighted t-Normalized Satisfiability problem is the version of SAT parameterized by k, where we are asked to find a satisfying assignment of weight exactly k, while the input formulas are t-normalized. For each $t \geq 2$, Weighted t-Normalized Satisfiability is W[t]-complete. It is furthermore FPT=W[0], while every class W[i] is a subset of W[$i+1$] and it is conjectured that inclusions are proper. The classes in the W-hierarchy are also closed under parameterized reductions.

For a parameterized problem with parameter k, an *FPT approximation scheme* (FPT-AS) is an algorithm which, for any $\epsilon > 0$, runs in time $O^*(f(k, \frac{1}{\epsilon}))$ (i.e. FPT time when parameterized by $k + \frac{1}{\epsilon}$) and produces a solution at most a multiplicative factor $(1+\epsilon)$ from the optimal. We will present approximation schemes with running-times of the form $(\log n/\epsilon)^{O(k)}$. These can be seen to imply an FPT running-time by the following well-known "win-win" argument:

Lemma 2. *If a parameterized problem with parameter k admits, for some $\epsilon > 0$, an algorithm running in time $O^*((\log n/\epsilon)^{O(k)})$, then it also admits an algorithm running in time $O^*((k/\epsilon)^{O(k)})$.*

Proof. We consider two cases: if $k \leq \sqrt{\log n}$ then $(\log n/\epsilon)^{O(k)} = (1/\epsilon)^{O(k)}(\log n)^{O(\sqrt{\log n})} = O^*((1/\epsilon)^{O(k)})$. If on the other hand, $k > \sqrt{\log n}$, we have $\log n \leq k^2$, so $O^*((\log n/\epsilon)^{O(k)}) = O^*((k/\epsilon)^{O(k)})$. □

Graphs

A *graph* is a pair $G = (V, E)$, where V is a set of idealized abstract objects referred to as the *vertices*, and E is a set of *edges*, or pairs of vertices. The edges of a graph define a symmetric relation on the vertices, the *adjacency* relation. Two adjacent vertices are also called *neighbors*. For a graph $G = (V, E)$, $n = |V|$ commonly denotes the number of vertices, $m = |E|$ the number of edges and we also let $V(G) \coloneqq V$ and $E(G) \coloneqq E$. For an edge $e = (u, v) = (v, u)$, the vertices v, u are its *endpoints* and are said to be *incident* on e. If the graph is *directed* its edges are generally called *arcs* and they are now ordered pairs of endpoints (with *tails* before *heads*).[1] The *degree* of a vertex v is the number of edges

[1] Excluding a brief foray in Section 3.4, we only consider undirected graphs here.

that v is incident on (the number of its neighbors) and is denoted by $\delta(v)$. For a subset $X \subseteq V$, we denote by $G[X]$ the graph *induced* by X, that is the graph whose vertex set is X and whose edge set consists of all of the edges in E that have both endpoints in X.

A *walk* of length l is a non-empty sequence $x_0, \ldots, x_l$ of vertices, each consequent pair x_i, x_{i+1} of which (for $i \in [0, l-1]$) is connected by an edge $(x_i, x_{i+1}) \in E$. A walk is *closed* if $x_0 = x_l$, while a *path* is a walk where no two vertices appear twice. A *cycle* is a closed walk where no two vertices appear twice, apart from the endpoints. A *connected component* of a graph is a subgraph in which there is at least one path between any pair of vertices, and which is connected to no additional vertices in the supergraph. The graph is *connected* if it consists of only one connected component. A subset $S \subset V$ is a *separator* for non-adjacent vertices u, v if the removal of S from G separates u and v into distinct connected components. A *tree* is a graph in which any two vertices are connected by exactly one path, or, equivalently, a connected acyclic graph, i.e. one that contains no cycles. A *forest* is a disjoint union of trees.

We denote by $d_G(v, u)$ the shortest-path *distance* from v to u in G, that is the minimum length of a path in G with endpoints v, u. We may omit subscript G if it is clear from the context. The maximum distance between vertices is the *diameter* of the graph, while the minimum among all the maximum distances between a vertex to all other vertices (their *eccentricities*) is considered as the *radius* of the graph. For a vertex v, we let $N_G^d(v)$ denote the (open) *d-neighborhood* of v in G, i.e. the set of vertices at distance $\leq d$ from v in G (without v), while for a subset $U \subseteq V$, $N_G^d(U)$ denotes the union of the d-neighborhoods of vertices $u \in U$. In a graph G whose maximum degree is bounded by Δ, the size of the d-neighborhood of any vertex v is upper bounded by the well-known *Moore bound*: $|N_G^d(v)| \leq \Delta \sum_{i=0}^{d} (\Delta - 1)^i$. For an integer q, the *q-th power graph of G*, denoted by G^q, is defined as the graph obtained from G by adding to $E(G)$ all edges between vertices $v, u \in V(G)$ for which $d_G(v, u) \leq q$.

Two vertices u, v are *independent* if there is no edge between them, or $(u, v) \notin E$. Similarly, a set of vertices is independent if every pair of its vertices is independent (also called a *stable set*). Conversely, a set of vertices is a *clique* if every pair of its vertices is connected by an edge. A *bipartite* graph $G = (A \cup B, E)$ is a graph whose vertex set is divided into two independent sets A, B and if every vertex of A is connected to every vertex of B the graph is called a *bi-clique*. We let K_n denote a clique on n vertices (otherwise known as a *complete* graph) and K_{n_1,n_2} the bi-clique where $|A| = n_1$ and $|B| = n_2$. It is well-known that a graph is bipartite if and only if it contains no odd-length cycles. A graph is *r-regular* if all its vertices are of degree r and specifically *cubic* when $r = 3$. In a *chordal* graph all cycles on ≥ 4 vertices have a *chord*, i.e. an edge connecting two vertices of the cycle, that is not part of the cycle.

A *planar* graph can be embedded in the (Euclidean) plane in such a way that its edges intersect only at their endpoints. An *edge-weighted* graph has a *weight function* $w : E \to \mathbb{N}^+$ associated with it that defines the *length* of each edge. All above definitions on distance can be extended to accommodate edge weights. The *triangle inequality* here requires that $d(u, v) + d(v, w) \geq d(u, w)$ for any u, v, w, where the distance function d obeys the weight function w (which is thus a *metric*).

For more information on these concepts the reader is referred to the standard textbooks: for classical complexity theory see [5, 49, 87, 90], for graph-theoretical notions [16, 19, 39], for approximation algorithms [95, 96], for parameterized complexity [35, 40, 47, 79] and specifically for the ETH [60, 61].

Parameters

Treewidth and *pathwidth* are standard notions in parameterized complexity that measure how close a graph is to being a tree or path (see [10, 11, 14, 71]). Due to their similarity, we focus on treewidth here and only refer to pathwidth when required. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$ with $T = (I, F)$ a tree and $\mathcal{X} = \{X_i | i \in I\}$ a family of subsets of V (called *bags*), one for each node of T, with the following properties:

1) $\bigcup_{i \in I} X_i = V$;

2) for all edges $(v, w) \in E$, there exists an $i \in I$ with $v, w \in X_i$;

3) for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree decomposition $((I, F), \{X_i | i \in I\})$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G, denoted by $\mathrm{tw}(G)$.

Moreover, for rooted T, let $G_i = (V_i, E_i)$ denote the *terminal subgraph* defined by node $i \in I$, i.e. the induced subgraph of G on all vertices in bag i and its descendants in T. Also let $N_i(v)$ denote the neighborhood of vertex v in G_i and $d_i(u, v)$ denote the distance between vertices u and v in G_i, while $d(u, v)$ (absence of subscript) remains the distance in G. *Path decompositions* and *pathwidth* are similarly defined, with the difference of T being a path instead of a tree.

In addition, a tree decomposition can be converted to a *nice* tree decomposition of the same width (in $O(\mathrm{tw}^2 \cdot n)$ time and with $O(\mathrm{tw} \cdot n)$ nodes): the tree here is rooted and binary, while nodes can be of four types:

a) Leaf nodes i are leaves of T and have $|X_i| = 1$;

b) Introduce nodes i have one child j with $X_i = X_j \cup \{v\}$ for some vertex $v \in V$ and are said to *introduce* v;

c) Forget nodes i have one child j with $X_i = X_j \setminus \{v\}$ for some vertex $v \in V$ and are said to *forget* v;

d) Join nodes i have two children denoted by $i - 1$ and $i - 2$, with $X_i = X_{i-1} = X_{i-2}$.

Nice tree decompositions were introduced by Kloks in [71] and using them does not in general give any additional algorithmic possibilities, yet algorithm design becomes considerably easier. See Figure 2.1 for examples of tree decompositions.

We will also make use of the notion of *clique-width* (see [33]): the set of graphs of clique-width cw is the set of vertex-labelled graphs that can be inductively constructed by using the following operations:

1) Introduce: $i(l)$, for $l \in [1, \mathrm{cw}]$ is the graph consisting of a single vertex with label l;

2) Join: $\eta(G, a, b)$, for G having cliquewidth cw and $a, b \in [1, \mathrm{cw}]$ is the graph obtained from G by adding all possible edges between vertices of label a and vertices of label b;

3) Rename: $\rho(G, a, b)$, for G having cliquewidth cw and $a, b \in [1, \mathrm{cw}]$ is the graph obtained from G by changing the label of all vertices of label a to b;

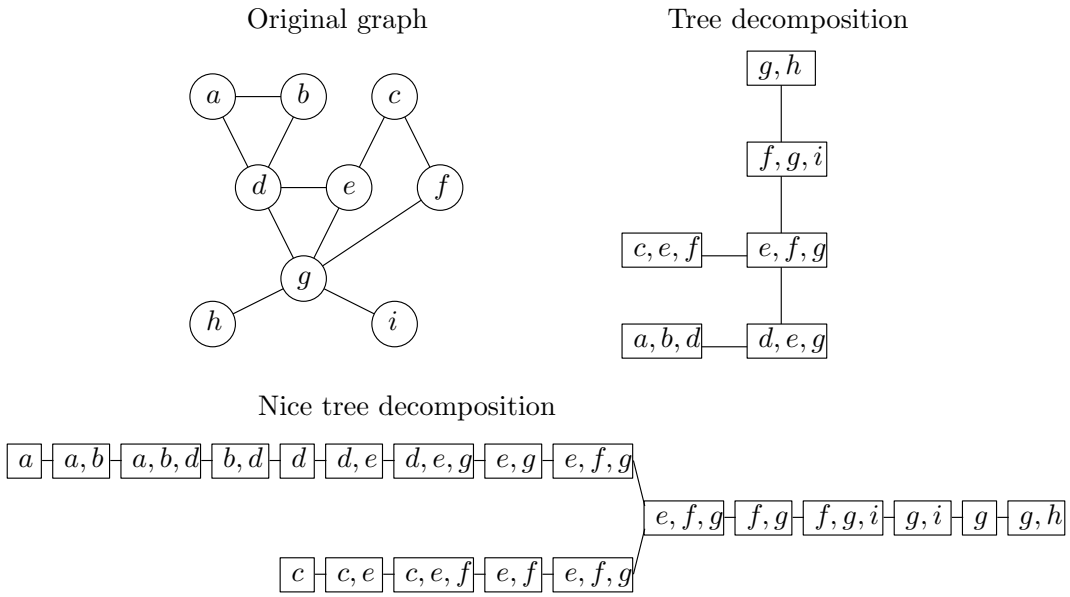


Figure 2.1: Example tree decompositions for a small graph of tw = 2.

4) Union: $G_1 \cup G_2$ (or $G_1 \otimes G_2$), for G_1, G_2 having cliquewidth cw is the disjoint union of graphs G_1, G_2.

Note we here assume the labels are integers in $[1, \mathrm{cw}]$, for ease of exposition.

A *clique-width expression* of width cw for $G = (V, E)$ is a recipe for constructing a cw-labelled graph isomorphic to G. More formally, a clique-width expression is a rooted binary tree T_G, such that each node $t \in T_G$ has one of four possible types, corresponding to the operations given above. In addition, all leaves are introduce nodes, each introduce node has a label associated with it and each join or rename node has two labels associated with it. For each node t, the graph G_t is defined as the graph obtained by applying the operation of node t to the graph (or graphs) associated with its child (or children). All graphs G_t are subgraphs of G and for all leaves of label l, their associated graph is $i(l)$. See Figure 2.2 for a small example.

Additionally, we will use the parameters *vertex cover number* and *feedback vertex set number* of a graph G, which are the sizes of the minimum vertex set whose deletion leaves the graph edgeless, or acyclic, respectively. Finally, we will consider the related notion of *tree-depth* [83], which is defined as the minimum height of a rooted forest whose completion (the graph obtained by connecting each node to all its ancestors) contains the input graph as a subgraph. Intuitively, where treewidth measures how far a graph is from being a tree, tree-depth measures how far a graph is from being a star.

We will denote these parameters for a graph G as $\mathrm{tw}(G), \mathrm{pw}(G), \mathrm{cw}(G), \mathrm{vc}(G), \mathrm{fvs}(G)$, and $\mathrm{td}(G)$, and will omit G if it is clear from the context. We also note (avoiding detailed definitions) that, alternatively, the treewidth $\mathrm{tw}(G)$, pathwidth $\mathrm{pw}(G)$, tree-depth $\mathrm{td}(G)$ and vertex cover $\mathrm{vc}(G)$ of a graph G can be defined as the minimum of the maximum clique-size (-1 for $\mathrm{tw}(G)$, $\mathrm{pw}(G)$ and $\mathrm{vc}(G)$) among all supergraphs of G that are of type chordal, interval, trivially perfect and threshold, respectively. We recall the well-known relations between these parameters, justifying the hierarchy given in Figure 2.3:

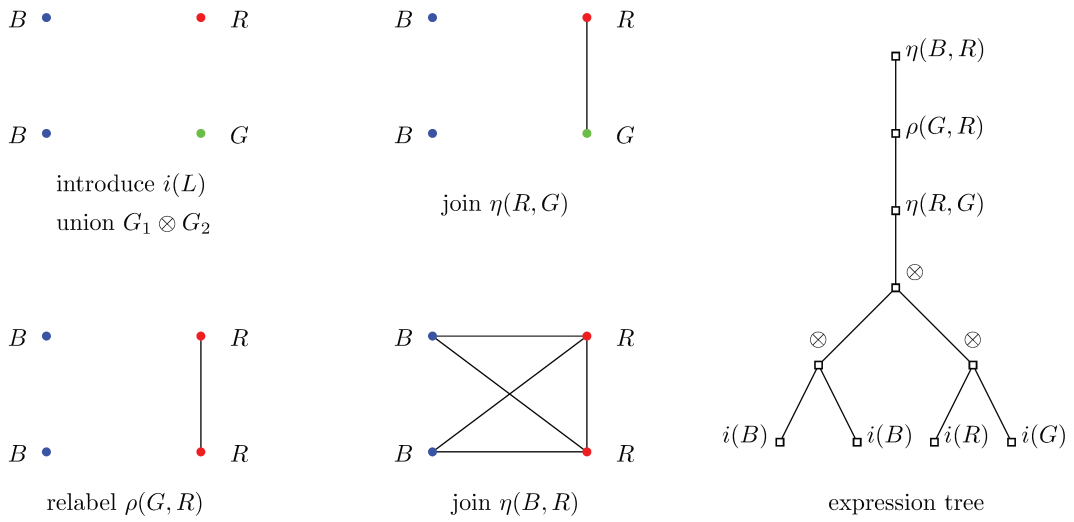


Figure 2.2: The construction of a simple graph using the clique-width operations.

Lemma 3. *[12, 34] For any graph G we have $tw(G) \leq pw(G) \leq td(G) \leq vc(G)$, $tw(G) \leq fvs(G) \leq vc(G)$, $cw(G) \leq pw(G) + 1$, and $cw(G) \leq 2^{tw(G)+1} + 1$.*

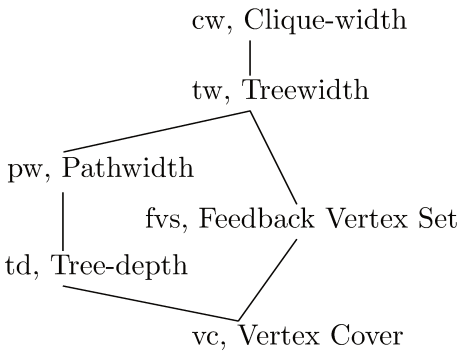


Figure 2.3: Relationships between parameters under consideration. Parameter size increases and algorithmic results are inherited in the downwards direction (from cw to vc), while hardness results are inherited upwards (from vc to cw).

Additionally, we will require the equivalent definition of pathwidth via the *mixed search number* $\mathrm{ms}(G)$ (see [92]). In a *mixed search game*, a graph G is considered as a system of tunnels. Initially, all edges are contaminated by a gas and an edge is *cleared* by placing searchers at both its endpoints simultaneously or by sliding a searcher along the edge. A cleared edge is re-contaminated if there is a path from a contaminated 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 G, denoted by $\mathrm{ms}(G)$, is the minimum number of searchers required for a winning strategy of mixed searching on G.

The following lemma from [92] shows the relationship between $\mathrm{ms}(G)$ and $\mathrm{pw}(G)$:

Lemma 4. *[92] For a graph G, it is $pw(G) \leq ms(G) \leq pw(G) + 1$.*

Notation

We use $\log(n), \ln(n)$ to denote the base-2 and natural logarithms of n, respectively, while $\log_{1+\delta}(n)$ is the logarithm base-$(1+\delta)$, for $\delta > 0$. Recall also that $\log_{1+\delta}(n) = \log(n)/\log(1+\delta)$. The functions $\lfloor x \rfloor$ and $\lceil x \rceil$, for $x \in \mathbb{R}$, denote the maximum/minimum integer that is not larger/smaller than x, respectively.

2.2 Problems and state-of-the-art

Covering problems

We begin with the definitions of the well-known Dominating Set and Set Cover problems. As here we focus on graph-theoretical formulations, we will mostly consider Dominating Set.

Definition 5. *In the* Dominating Set *problem we are given an undirected graph* $G = (V, E)$ *and an integer* k *and are asked to find a subset of vertices* $D \subseteq V$*, with* $|D| \leq k$*, such that every vertex not in* D *has at least one neighbor in* D*:* $\forall v \notin D : N(v) \cap D \neq \emptyset$*.*

Definition 6. *Given a* universe $\mathcal{U} = \{u_1, \ldots, u_n\}$ *of* elements *and a family* $\mathcal{S} = \{S_1, \ldots, S_m\}$ *of* subsets *of* $\mathcal{U}$*, a* cover *is a subfamily* $\mathcal{C} \subseteq \mathcal{S}$ *of sets whose union is* $\mathcal{U}$*. In* Set Cover*, the input is a pair* $\mathcal{U}, \mathcal{S}$ *and an integer* k*; the question is whether there is a set cover of size* k *or less.*

The Dominating Set problem is NP-complete [49] and cannot be approximated by a factor better than $\ln n$ [82]. It is also W[2]-complete parameterized by the size of an optimal solution k and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable f under the ETH [35]. Concerning approximability of k in FPT-time, it is shown in [29] that any constant-approximation of the parameterized by k Dominating Set problem is W[1]-hard. Furthermore, the existence of a $f(\gamma(G)) \cdot |G|^{(log\ \gamma(G))^{\varepsilon/12}}$-time algorithm is also ruled out, where $\gamma(G)$ denotes the size of the minimum and assuming the ETH, which on every input graph G outputs a dominating set of size at most $\sqrt[3+\varepsilon]{log\ (\gamma(G))} \cdot \gamma(G)$ for every $0 < \varepsilon < 1$. Subsequently, it was shown in [25] that under the Gap-ETH,[2] no $F(k)$-approximation FPT-time algorithm for Dominating Set parameterized by k exists for any computable function F. Finally, providing an improvement in terms of complexity assumptions, [89] shows there is no $F(k)$-approximation algorithm for the problem: (a) in FPT-time under FPT$\neq$W[1], (b) in $T(k)n^{o(k)}$-time under the ETH, (c) in $T(k)n^{k-\epsilon}$-time for every integer $k \geq 2$, under the SETH.

Concerning structural parameters, the problem is solvable in time $O^*(4^{\text{cw}})$ [15] and thus FPT when parameterized by any of the structural parameters we consider here. Specifically for treewidth, a series of papers had culminated into an $O^*(3^{\text{tw}})$ algorithm [93, 2, 94], while on the other hand, [76] showed that an $O^*((3-\epsilon)^{\text{pw}})$ algorithm would violate the SETH, where pw denotes the input graph's pathwidth. Further, [84] notes that the lower bound for pathwidth/treewidth would also imply no $(3-\epsilon)^{\text{cw}} \cdot n^{O(1)}$-time algorithm exists for clique-width under the SETH as well, since clique-width is at most 1 larger than pathwidth.

The (k, r)-Center problem is a generalization of Dominating Set (for $r = 1$):

[2] A stronger assumption than the ETH, stating that no subexponential-time algorithm can distinguish between satisfiable 3-SAT instances and those that are not even $(1-\epsilon)$-satisfiable for some $\epsilon > 0$.

Definition 7. *In* (k, r)-Center *we are given a graph* $G = (V, E)$ *and a weight function* $w : E \mapsto \mathbb{N}^+$ *which satisfies the triangle inequality and defines the length of each edge and are asked if there exists a set* K *(the* center-set*) of at most* k *vertices of* V*, so that* $\forall u \in V \setminus K$ *we have* $\min_{v \in K} d(v, u) \leq r$*, where* $d(v, u)$ *denotes the shortest-path distance from* v *to* u *under weight function* w*.*

If w assigns weight 1 to all edges we say that we have an instance of *unweighted* (k, r)-Center. Dealing with (k, r)-Center we allow, in general, the weight function w to be non-symmetric. We also require edge weights to be strictly positive integers but, as we will see (Lemma 43), this is not a significant restriction. We will say that a vertex is *covered* (resp. *dominated* for $r = 1$) by a center-set K if it is at distance $\leq r$ from a vertex $u \in K$.

Hardness of (k, r)-Center is inherited from Dominating Set. Moreover, the optimal r cannot be approximated in polynomial time by a factor better than 2, even on planar graphs [45]. In [22], generalizing the previously mentioned results for Dominating Set, it is shown that (k, r)-Center can be solved in $O^*((2r + 1)^{\text{tw}})$ (already implied by the results of [50]), but not faster assuming the SETH, while its connected variant (i.e. where solutions must consist of connected subgraphs) can be solved in $O^*((2r + 2)^{\text{tw}})$, but not faster.

For the edge-weighted variant and (unrelated) parameters, [46] shows that a $(2 - \epsilon)$-approximation is W[2]-hard for parameter k and NP-hard for graphs of *highway dimension* $h = O(\log^2 n)$, while also offering a 3/2-approximation algorithm of running-time $2^{O(kh \log(h))} \cdot n^{O(1)}$, exploiting the similarity of this problem with that of solving Dominating Set on graphs of bounded vc. For unweighted graphs, [75] provides efficient (linear/polynomial) algorithms computing $(r + O(\mu))$-dominating sets and $+O(\mu)$-approximations for (k, r)-Center, where μ is the *tree-breadth* or *cluster diameter* in a *layering partition* of the input graph, while [41] gives a polynomial-time bicriteria approximation scheme for graphs of bounded *genus*.

Packing problems

We next give the definitions of the well-known Independent Set (IS) and Set Packing problems. Due to our interest in graph-theoretical formulations we mostly consider Independent Set.

Definition 8. *In the* Independent Set *problem we are given a graph* $G = (V, E)$ *and an integer* k *and are asked to find a subset of vertices* $I \subseteq V$*, with* $|I| \geq k$*, such that every pair of vertices in* I *is* independent*, i.e. there is no edge between them:* $\forall u, v \in I : (u, v) \notin E$*.*

Definition 9. *In the* Set Packing *problem we are given an integer* k*, a* universe $\mathcal{U} = \{u_1, \ldots, u_n\}$ *of* elements *and a family* $\mathcal{S} = \{S_1, \ldots, S_m\}$ *of* subsets *of* $\mathcal{U}$*, and are asked to determine if there is a subfamily* $S \subseteq \mathcal{S}$ *of subsets (a* packing*), such that all sets in* S *are pairwise disjoint, while the size of the packing is* $|S| \geq k$*.*

The Independent Set problem is NP-complete [49] and inapproximable in polynomial time within $n^{1-\epsilon}$ for any $\epsilon > 0$ [58, 97], but admits a PTAS for planar graphs [6]. For graphs of degree bounded by $\Delta \geq 3$, Independent Set is APX-complete [86] and not approximable in polynomial time to Δ^ϵ for some $\epsilon > 0$ [3], but there are greedy $(\Delta + 2)/3$-approximations [57]. For super-polynomial running-times, any ρ-approximation for the Independent Set problem must require time at least $2^{n^{1-\epsilon}/\rho^{1+\epsilon}}$, almost matching the upper bound of $2^{n/r}$ [26, 36].

For a graph G, we let $\alpha(G)$ denote its *independence number*, i.e. the size of its largest independent set. We also recall here the following result by [26] that some of our later reductions will be relying on (slightly paraphrased, see also [21]), that can be seen as implying the (randomized) $\Delta^{1-\epsilon}$-inapproximability of Independent Set in polynomial time:

Theorem 10 ([26], Theorem 5.2)**.** *For any sufficiently small $\epsilon > 0$ and any $\Delta \leq N^{5+O(\epsilon)}$, there is a randomized polynomial-time reduction that builds from a formula ϕ of* SAT *on N variables a graph G of size $n = N^{1+\epsilon}\Delta^{1+\epsilon}$ and maximum degree Δ, such that with high probability:*

- *If ϕ is satisfiable, then $\alpha(G) \geq N^{1+\epsilon}\Delta$;*
- *If ϕ is not satisfiable, then $\alpha(G) \leq N^{1+\epsilon}\Delta^{2\epsilon}$.*

The standard parameterization of the problem by solution size k is W[1]-complete and cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable f under the ETH [35]. Furthermore, [25] shows there is no FPT-time $o(k)$-approximation, again under the Gap-ETH, improving upon the constant-inapproximability of the problem in FPT-time given in [20, 55]. For all structural parameters considered here the problem is FPT, as there is an $O^*(2^{\text{cw}})$-time algorithm [53]. Specifically for treewidth, the $O^*(2^{\text{tw}})$-time algorithm is complemented by a matching SETH-based lower bound of $(2-\epsilon)^{\text{tw}} \cdot n^{O(1)}$ [35, 76, 94].

As before, the d-Scattered Set problem is a generalization of Independent Set.

Definition 11. *In the d-*Scattered Set *problem, we are given graph $G = (V, E)$ and a metric weight function $w : E \mapsto \mathbb{N}^+$ that gives the length of each edge and are asked if there exists a set K of at least k* selections *from V, such that the distance between any pair $v, u \in K$ is at least $d(v, u) \geq d$, where $d(v, u)$ denotes the shortest-path distance from v to u under weight function w.*

If w assigns weight 1 to all edges, the variant is also called *unweighted*. We will also denote by $OPT_d(G)$ the maximum size of a d-scattered set in G, thus $\alpha(G) = OPT_2(G)$.

The problem is APX-hard for $r, d \geq 3$ on r-regular graphs, while there are polynomial-time $O(r^{d-1})$-approximations [43]. The same paper also shows a polynomial-time 2-approximation on cubic graphs and a PTAS for planar graphs and every fixed constant $d \geq 3$, extending the algorithm of [6] for Independent Set. For a class of graphs with at most a polynomial (in n) number of minimal separators (containing chordal graphs), d-Scattered Set can be solved in polynomial time for even d, while it remains NP-hard on chordal graphs and any odd $d \geq 3$ [81] (see also [42]).

For the odd values of d, a polynomial $\sqrt{n}$-approximation is given in [56]. This algorithm is in fact a corollary of a $\sqrt{n}$-approximation for Set Packing, since there is a correspondence between the d-Scattered Set problem for odd values of d and the formulation of independence based on sets [56]: a *strong stable set* corresponds to a 3-scattered set and is also known as a *2-packing*. Given a graph $G = (V, E)$ we can construct a set system $(\mathcal{U} = V, \mathcal{S} = \{N(v) \cup \{v\} : v \in V\})$, where the universe consists of the vertices and the sets are defined based on the *closed* neighborhood of each vertex v, i.e. that also includes v. In this way a strong stable set corresponds to a set of vertices whose closed neighborhoods have no overlap (being thus at distance $d \geq 3$), i.e. a set packing of $(\mathcal{U}, \mathcal{S})$. Note that this is also equivalent to an independent set in the *square* graph G^2, while in general, the $(2q+1)$-Scattered Set problem is equivalent to Independent Set in $(G^q)^2$.

For bipartite graphs, d-Scattered Set is NP-hard to approximate within a factor of $n^{1/2-\epsilon}$ and W[1]-hard for any fixed $d \geq 3$ [42]. The problem is moreover shown to be NP-hard even for planar bipartite graphs of maximum degree 3, while a 1.875-approximation is available on cubic graphs [44]. Furthermore, [48] gives an EPTAS on (apex)-minor-free graphs, based on the theory of bidimensionality, while on a related result an $n^{O(\sqrt{n})}$-time algorithm exists for planar graphs, making use of Voronoi diagrams and based on ideas previously used to obtain geometric QPTASs (i.e. *quasi-polynomial-time approximation schemes*, being exponential in poly-logarithmic functions) [80]. Finally, [88] shows that it admits an almost linear *kernel* (intuitively, the difficult part of a parameterized problem) on every *nowhere dense* graph class, being a common generalization of several classes including graphs of bounded degree.

In our parameterized reductions we will also make use of k-Multicolored Independent Set, a well-known W[1]-complete variant that is also unsolvable in time $f(k) \cdot n^{o(k)}$ under the ETH [35].

Definition 12. *In k-Multicolored Independent Set, we are given a graph $G = (V, E)$, with V partitioned into k cliques $V = V_1 \uplus \cdots \uplus V_k$, $|V_i| = n, \forall i \in [1, k]$, and are asked to find an $S \subseteq V$, such that $G[S]$ forms an independent set and $|S \cap V_i| = 1, \forall i \in [1, k]$.*

3

On the Structurally Parameterized (k, r)-Center problem

In this chapter we study the (k, r)-Center problem. It is an extremely well-investigated optimization problem with numerous applications. It has a long history, especially from the point of view of approximation algorithms, where the objective is typically to minimize r for a given k [1, 41, 45, 59, 69, 70, 72, 85, 95]. The converse objective (minimizing k for a given r) has also been well-studied, with the problem being typically called r-Dominating Set in this case [24, 30, 50, 77, 91].

Because (k, r)-Center generalizes Dominating Set (which corresponds to the case $r = 1$), the problem can already be seen to be hard, even to approximate (under standard complexity assumptions). Since this hardness persists when considering the standard parameterization of the problem with non-trivial parameterized approximations also precluded, we are strongly motivated to investigate the problem's complexity when the input graph has some restricted structure.

Our results: Our goal is to perform a complete analysis of the complexity of (k, r)-Center that takes into account this input structure by using the framework of parameterized complexity. In particular, we provide *fine-grained* upper and lower bound results on the complexity of (k, r)-Center with respect to the most widely studied parameters that measure a graph's structure: treewidth **tw**, clique-width **cw**, tree-depth **td**, vertex cover **vc**, and feedback vertex set **fvs**. In addition to the intrinsic value of determining the precise complexity of (k, r)-Center, this approach is further motivated by the fact that FPT algorithms for this problem have often been used as building blocks for more elaborate approximation algorithms [37, 41]. Indeed, (some of) these questions have already been considered, but we provide a number of new results that build on and improve the current state-of-the-art. Along the way, we also close a gap on the complexity of the flagship Dominating Set problem parameterized by clique-width. Specifically, we prove the following:

- (k, r)-Center can be solved (on unweighted graphs) in time $O^*((3r + 1)^{\mathrm{cw}})$ (if a clique-width expression is supplied with the input), but it cannot be solved in time $O^*((3r + 1 - \epsilon)^{\mathrm{cw}})$ for any fixed $r \geq 1$, unless the SETH fails.

  The algorithmic result relies on standard techniques (dynamic programming on clique-width expressions, fast subset convolution), as well as several problem-specific observations which are required to obtain the desired table size. The SETH lower

bound follows from a direct reduction from SAT. A noteworthy consequence of our lower bound result is that, for the case of Dominating Set, it closes the gap between the complexity of the best known algorithm ($O^*(4^{\mathrm{cw}})$ [15]) and the best previously known lower bound ($O^*((3-\epsilon)^{\mathrm{cw}})$ [76]).

- (k,r)-Center cannot be solved in time $n^{o(\mathrm{vc}+k)}$ on edge-weighted graphs, or time $n^{o(\mathrm{fvs}+k)}$ on unweighted graphs, unless the ETH is false.

  It was already known that an FPT algorithm parameterized just by tw (for unbounded r) is unlikely to be possible [22]. These results show that the same holds for the two more restrictive parameters fvs and vc, even if k is also added as a parameter. They are (asymptotically) tight, since it is easy to obtain $O^*(n^{\mathrm{fvs}})$, $O^*(n^{\mathrm{vc}})$, and $O^*(n^k)$ algorithms. We remark that (k,r)-Center is a rare example of a problem that turns out to be hard parameterized by vc. We complement these lower bounds by an FPT algorithm for the unweighted case, running in time $O^*(5^{\mathrm{vc}})$.

- (k,r)-Center can be solved in time $O^*(2^{O(\mathrm{td}^2)})$ for unweighted graphs, but if it can be solved in time $O^*(2^{o(\mathrm{td}^2)})$, then the ETH is false.

  Here the upper bound follows from known connections between a graph's tree-depth and its diameter, while the lower bound follows from a reduction from 3-SAT. We remark that this is a somewhat uncommon example of a parameterized problem whose parameter dependence turns out to be exponential in the *square* of the parameter.

The results above, together with the recent work of [22] which showed tight bounds of $O^*((2r+1)^{\mathrm{tw}})$ regarding the problem's complexity parameterized by tw (see also [50]), give a complete, and often fine-grained, picture on (k,r)-Center for the most important graph parameters. One of the conclusions that can be drawn is that, as a consequence of the problem's hardness for vc (in the weighted case) and fvs, there are few cases where we can hope to obtain an FPT algorithm without bounding the value of r. In other words, as r increases the complexity of exactly solving the problem quickly degenerates away from the case of Dominating Set, which is FPT for all considered parameters.

A further contribution of this chapter is to complement this negative view by pointing out that it only applies if one insists on solving the problem *exactly*. If we allow algorithms that return a $(1+\epsilon)$-approximation to the optimal r, for arbitrarily small $\epsilon > 0$ and while respecting the given value of k, we obtain the following:

- There exist algorithms which, for any $\epsilon > 0$, when given a graph that admits a (k,r)-center, return a $(k,(1+\epsilon)r)$-center in time $O^*((tw/\epsilon)^{O(\mathrm{tw})})$, or $O^*((cw/\epsilon)^{O(\mathrm{cw})})$, assuming a tree decomposition or clique-width expression is given in the input.

The tw approximation algorithm is based on a technique introduced in [73], while the cw algorithm relies on a new extension of an idea from [54], which may be of independent interest. Thanks to these approximation algorithms, we arrive at an improved understanding of the complexity of (k,r)-Center by including the question of approximation, and obtain algorithms which continue to work efficiently even for large values of r. Figure 3.1 illustrates the relationships between parameters and Table 3.1 summarizes our results.

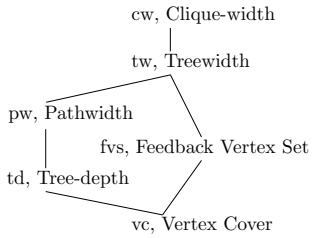


Figure 3.1: Relationships of parameters. Algorithmic results are inherited downwards, hardness results upwards.

	cw	tw	fvs	td	vc
FPT exact	24 (w/u)	46 (w/u)		57 (u)	29 (u)
FPT-AS	45 (w/u)	42 (w/u)			
SETH LB	21 (u)				
ETH LB			28 (w/u)	35 (u)	27 (w)
W[1]-hard			28 (w/u)		27 (w)

Table 3.1: A summary of our results (theorem numbers) for all considered parameters. Initials u/w denote the unweighted/weighted variants of the problem.

3.1 Clique-width

3.1.1 Lower bound based on the SETH

In this subsection we prove that for any fixed constant $r \geq 1$, the existence of any algorithm for (k, r)-Center of running-time $O^*((3r + 1 - \epsilon)^{\text{cw}})$, for some $\epsilon > 0$, would imply the existence of some algorithm for SAT of running-time $O^*((2 - \delta)^n)$, for some $\delta > 0$.

Before we proceed, let us recall the high-level idea behind the SETH lower bound for Dominating Set given in [76], as well its generalization to (k, r)-Center given in [22]. In both cases the key to the reduction is the construction of long paths, which are conceptually divided into blocks of $2r + 1$ vertices. The intended solution consists of selecting, say, the i-th vertex of a block of a path, and repeating this selection in all blocks of this path. This allows us to encode $(2r+1)^t$ choices, where t is the number of paths we make, which ends up being roughly equal to the treewidth of the construction.

The reason this construction works in the converse direction is that, even though the optimal (k, r)-Center solution may "cheat" by selecting the i-th vertex of a block, and then the j-th vertex of the next, one can see that we must have $j \leq i$. Hence, by making the paths that carry the solution's encoding long enough we can ensure that the solution eventually settles into a pattern that encodes an assignment to the original formula (which can be "read" with appropriate gadgets).

In our lower bound construction for clique-width we need to be able to "pack" more information per unit of width: instead of encoding $(2r+1)$ choices for each unit of treewidth, we need to encode $(3r + 1)$ choices for each label. Our high-level plan to achieve this is to use *a pair* of long paths for each label. Because we only want to invest one label for each pair of paths we are forced to periodically (every $2r + 1$ vertices) add cross-edges between them, so that the connection between blocks can be performed with a single join operation. See the paths A_1, B_1 in Figure 3.7 for an illustration.

Our plan now is to encode a solution by selecting a pair of vertices that will be repeated in each block, for example every i-th vertex of A_1 and every j-th vertex of B_1. One may naively expect that this would allow us to encode $(2r + 1)^2$ choices for each label (which would lead to a SETH lower bound that would contradict the algorithm of Subsection 3.1.2). However, because of the cross-edges, the optimal (k, r)-Center solution is not as well-behaved on a pair of cross-connected paths as it was on a path, and this makes it much harder to execute the converse direction of the reduction. Our strategy will therefore be

to identify $(3r+1)$ canonical selection pairs, order them, and show that any valid solution must be well-behaved with respect to these pairs. We will then use these pairs to encode $(3r+1)^{\text{cw}}$ choices and still get the converse direction of the reduction to work (if all paths are sufficiently long).

We next describe the construction of a graph G, given some $\epsilon < 1$ and an instance ϕ of SAT with n variables and m clauses. We first choose an integer $p \geq \dfrac{1}{(1-\lambda)\log_2(3r+1)}$, for $\lambda = \log_{3r+1}(3r+1-\epsilon) < 1$, for reasons that become apparent in the proof of Theorem 21. Note that for the results of this subsection, both r and p are considered constants. We then group the variables of ϕ into $t = \lceil \frac{n}{\gamma} \rceil$ groups $F_1, \ldots, F_t$, for $\gamma = \lfloor \log_2(3r+1)^p \rfloor$, being also the maximum size of any such group. Our construction uses a main Block gadget $\hat{G}$ and three smaller gadgets $\hat{T}_N$, $\hat{X}_N$ and $\hat{U}_N$ as parts.

Guard gadget $\hat{T}_N$: This gadget has N *input* vertices and its purpose is to allow for any selection of a single input vertex as a center to cover all vertices within the gadget, while offering no paths of length $\leq r$ between the inputs. Thus if at least two guard gadgets are attached to a set of N input vertices, any minimum-sized center-set will select one of them to cover all vertices in the gadgets, without interfering with whether the other inputs are covered by some outside selection.

Construction and size/cw bounds for $\hat{T}_N$: Given vertices $v_1, \ldots, v_N$, we construct the gadget as follows: we first make $\lfloor \frac{r}{2} \rfloor$ vertices $u_i^1, \ldots, u_i^{\lfloor \frac{r}{2} \rfloor}$ forming a path for each v_i and we connect each u_i^1 to each v_i. We then make another path on $\lceil \frac{r}{2} \rceil$ vertices, called $w^1, \ldots, w^{\lceil \frac{r}{2} \rceil}$, and we make vertices $u_i^{\lfloor \frac{r}{2} \rfloor}$ adjacent to the starting vertex w^1 of this path for all $i \in [1, N]$. Finally, if r is even, we make all vertices $u_i^{\lfloor \frac{r}{2} \rfloor}$ into a clique, while for odd r the construction is already complete. Figure 3.2 provides an illustration. The number of vertices in the gadget is $|\hat{T}_N| = N\lfloor \frac{r}{2} \rfloor + \lceil \frac{r}{2} \rceil$, while the gadget can also be constructed by a clique-width expression using at most this number of labels, by handling each vertex as an individual label.

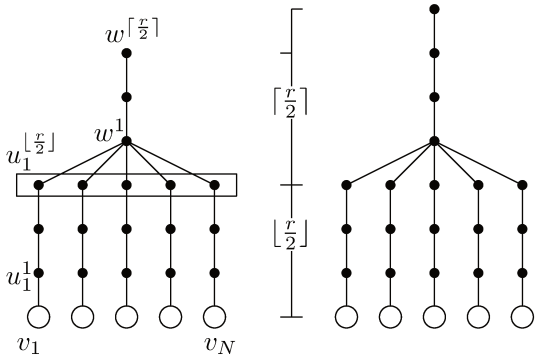


Figure 3.2: A general picture of the guard gadget $\hat{T}_N$ for even and odd r. Note the box indicating vertices forming a clique for the case of even r.

Lemma 13. *All vertices of gadget $\hat{T}_N$ are at distance $\leq r$ from any input vertex $v_i, i \in [1, N]$, while $d(v_i, v_j) = r+1$, for any $i \neq j \in [1, N]$.*

Proof. The distance from any input vertex v_i, with $i \in [1, N]$, to vertex $u_i^{\lfloor \frac{r}{2} \rfloor}$ is $d(v_i, u_i^{\lfloor \frac{r}{2} \rfloor}) = \lfloor \frac{r}{2} \rfloor$, while the distance from $u_i^{\lfloor \frac{r}{2} \rfloor}$ to $w^{\lceil \frac{r}{2} \rceil}$ is $d(u_i^{\lfloor \frac{r}{2} \rfloor}, w^{\lceil \frac{r}{2} \rceil}) = \lceil \frac{r}{2} \rceil$, thus $d(v_i, w^{\lceil \frac{r}{2} \rceil}) = \lfloor \frac{r}{2} \rfloor + \lceil \frac{r}{2} \rceil = r$. Further, the distance from $u_i^{\lfloor \frac{r}{2} \rfloor}$ to any vertex u_j^1, for $j \neq i$, is $d(u_i^{\lfloor \frac{r}{2} \rfloor}, u_j^1) = \lfloor \frac{r}{2} \rfloor$ for even r, and $d(u_i^{\lfloor \frac{r}{2} \rfloor}, u_j^1) = 1 + \lfloor \frac{r}{2} \rfloor$ for odd r. Thus $d(v_i, u_j^1) = \lfloor \frac{r}{2} \rfloor + \lfloor \frac{r}{2} \rfloor = r$ for even r, and $d(v_i, u_j^1) = \lfloor \frac{r}{2} \rfloor + 1 + \lfloor \frac{r}{2} \rfloor = r$ for odd r. Thus any vertex on these paths is at distance $\leq r$ from v_i, while any other input vertex v_j, being at distance 1 from its corresponding u_j^1, is at distance $r + 1$ from v_i. □

Clique gadget $\hat{X}_N$: This gadget again has N input vertices and the aim now is to make sure that any selection of a single input vertex as a center will cover all other vertices of the gadget, that no selection of any single vertex that is not an input would suffice instead, while all paths connecting the inputs are of distance exactly r.

Construction and size/cw bounds for $\hat{X}_N$: Given vertices $v_1, \ldots, v_N$, we first connect them to each other by paths on $r - 1$ new vertices, so that the distances between any two of them are exactly r. We then make all *middle* vertices that lie at distance $\lfloor \frac{r}{2} \rfloor$ from some vertex v_i on these paths adjacent to each other (into a clique): for even r the middle vertex of each path is at distance $r/2$ from the path's endpoint vertices v_i, v_j, while for odd r the middle vertices are the two vertices at distance $(r-1)/2$ from one of the path's endpoints v_i, v_j. Thus all vertices on these paths are at distance at most r from any vertex v_i, as they lie within distance $< \lfloor \frac{r}{2} \rfloor$ from a middle vertex on their path, which is at distance exactly $\lfloor \frac{r}{2} \rfloor + 1$ from any input vertex v_i. Next, we make another vertex $u_{i,j}^l$ for each middle vertex of these paths between v_i, v_j (and $l \in [1, 2]$ for odd r) and we make it adjacent to its corresponding middle vertex. We then add another vertex x that we also connect to all v_i vertices by paths using $r-1$ new vertices (thus at distance r from each v_i), making the middle vertices on these new paths adjacent to each other as well (into another, disjoint clique). See Figure 3.3 for an illustration. The size of the gadget is $|\hat{X}_N| = N + 1 + \binom{N}{2} \cdot r + N(r-1)$ for even r and $|\hat{X}_N| = N + 1 + \binom{N}{2} \cdot (r+1) + N(r-1)$ for odd r. The gadget can also be constructed by a clique-width expression using at most this number of labels, by handling each vertex as an individual label.

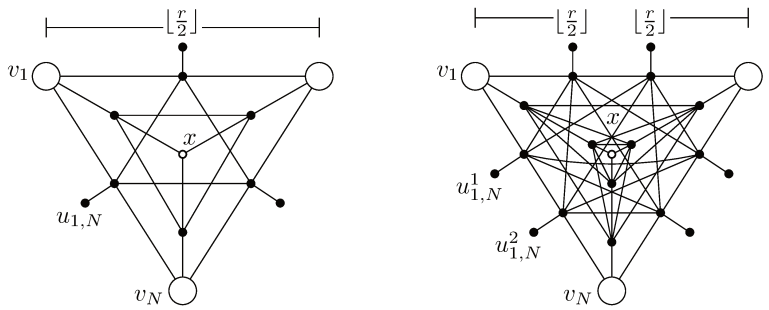


Figure 3.3: A general picture of the clique gadget $\hat{X}_N$ for even and odd r.

Lemma 14. *Any minimum-sized center-set restricted to the vertices of $\hat{X}_N$, will select exactly one of the input vertices $v_i, i \in [1, N]$, to cover all other vertices in $\hat{X}_N$.*

Proof. Assume the selection of any single vertex from $\hat{X}_N$ in any minimum-sized center-set: if the selected vertex is on one of the original paths, then vertex x is not covered, as its distance is at least $r+1$ from any such vertex (via the closest v_i), while if the selected vertex, say w, is on one of the new paths between some v_i and x, there will be at least one vertex $u^l_{j,k}$ on a path between two input vertices v_j, v_k for $l \in [1,2]$ that is not covered, as the distances from w to v_j, v_k are at least $1+\lfloor \frac{r}{2} \rfloor$ and the distances from there to the $u^l_{j,k}$ are also $1+\lfloor \frac{r}{2} \rfloor$. On the other hand, if the selected vertex is one of the inputs v_i, then all other vertices are covered: the distance from v_i to any other input v_j or x is exactly r, thus all vertices on these paths are covered (including vertices of the type $u^l_{i,j}$), while for vertices on paths not originating at v_i, the distance from v_i to some middle vertex on a path adjacent to it is $\lfloor \frac{r}{2} \rfloor$ and the distance from there to any vertex on some other path (or even adjacent to a middle vertex) is $\leq \lfloor \frac{r}{2} \rfloor$, giving an overall distance of $\leq r$. □

Assignment gadget $\hat{U}_N$: Once more, there are N input vertices $v_1, \ldots, v_N$ for this gadget, while the purpose here is to ensure that, assuming all input vertices have already been covered, any minimum-sized center-set will select exactly $N-1$ of them to cover all other vertices in the gadget.

Construction and size/cw bound for $\hat{U}_N$: We first connect all input vertices to each other by two distinct paths each containing r new vertices, so that all distances between any pair of input vertices are exactly $r+1$. Let the vertices on these paths between v_i, v_j be $u^l_{i,j}$ and $\hat{u}^l_{i,j}$ for $l \in [1,r]$. Then, for odd r, we also attach a path of $\lfloor \frac{r}{2} \rfloor$ vertices to the middle vertex of each path, that is, the vertices $u^{\lfloor \frac{r}{2} \rfloor+1}_{i,j}$ and $\hat{u}^{\lfloor \frac{r}{2} \rfloor+1}_{i,j}$ that are at distance $\lfloor \frac{r}{2} \rfloor+1$ from both endpoints v_i, v_j of their paths. We call the vertices on these new paths $w^m_{i,j}$ and $\hat{w}^m_{i,j}$ for $m \in [1, \lfloor \frac{r}{2} \rfloor]$. For even r, we make a vertex $w^{r/2}_{i,j}$ (resp. $\hat{w}^{r/2}_{i,j}$) for each path and attach two paths of $r/2-1$ vertices to it, naming the vertices on these paths $w^{o,m}_{i,j}$ (resp. $\hat{w}^{o,m}_{i,j}$) for $o \in [1,2]$ and $m \in [1, r/2-1]$, finally attaching the other endpoint vertex $w^{1,1}_{i,j}$ (resp. $\hat{w}^{1,1}_{i,j}$) to $u^{r/2}_{i,j}$ (resp. $\hat{u}^{r/2}_{i,j}$) and also $w^{2,1}_{i,j}$ (resp. $\hat{w}^{2,1}_{i,j}$) to $u^{r/2+1}_{i,j}$ (resp. $\hat{u}^{r/2+1}_{i,j}$), being the vertices at distance $r/2$ from one of the two endpoints v_i (and $r/2+1$ from the other v_j). Thus between any two inputs v_i, v_j, there are two vertices $w^{\lfloor \frac{r}{2} \rfloor}_{i,j}, \hat{w}^{\lfloor \frac{r}{2} \rfloor}_{i,j}$ at distance exactly r from both. See Figure 3.4 for an illustration. The size of the gadget is $|\hat{U}_N| = N + 2\binom{N}{2} \cdot (r + \lfloor \frac{r}{2} \rfloor)$ for odd r and $|\hat{U}_N| = N + 2\binom{N}{2} \cdot (2r-1)$ for even r, while the gadget can also be constructed by a clique-width expression using at most this number of labels, by handling each vertex as an individual label.

Lemma 15. *Assuming all input vertices $v_1, \ldots, v_N$ need not be covered by this selection, any minimum-sized center-set restricted to the vertices of $\hat{U}_N$ will select exactly $N-1$ of the input vertices to cover all other vertices in $\hat{U}_N$.*

Proof. The claim on the gadget's function is shown by induction on N: for all $N \geq 2$ any minimum-sized center-set will select exactly $N-1$ input vertices, while if any non-input vertex is selected there will be at least N vertices required. The base case is $N=2$ and we have two input vertices v_1, v_2 and two paths between them, with selection of either v_1 or v_2 indeed covering all vertices on these paths. On the other hand, as on each path between v_1, v_2 there is a vertex $w^{\lfloor \frac{r}{2} \rfloor}_{1,2}$ or $\hat{w}^{\lfloor \frac{r}{2} \rfloor}_{1,2}$ at distance exactly r from both v_1, v_2 (and the distance between them is $2r$), selection of any single vertex on these paths will cover all vertices on its path but not all vertices on the other path and thus at least two selections

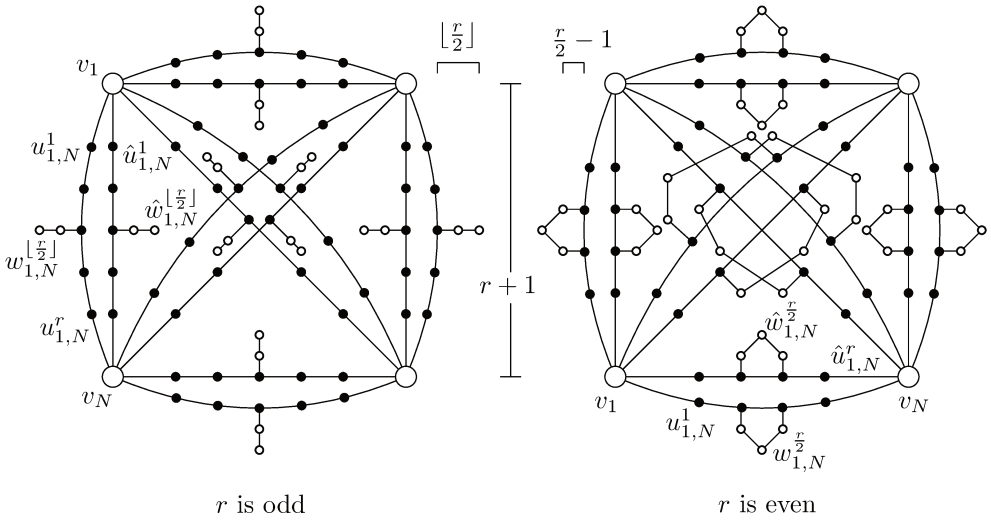


Figure 3.4: A general picture of the assignment gadget $\hat{U}_N$ for odd and even r.

will be required. For the induction step, assuming the claim holds for $N-1$, we extend it for N: the gadget $\hat{U}_N$ is constructed by the gadget $\hat{U}_{N-1}$ by adding a new input vertex v_N and two paths from it to every other input vertex $v_1, \ldots, v_{N-1}$. Let v_i with $i \in [1, N-1]$ be the vertex that is not selected from the original $N-1$ input vertices of the $\hat{U}_{N-1}$ gadget. Then in $\hat{U}_N$, all vertices that were already in $\hat{U}_{N-1}$ have been covered, as well as all vertices on the paths between vertices v_j with $j \in [1, N-1], j \neq i$ and v_N. The only vertices that still need to be covered are the ones on the two paths between v_i and v_N. Again, as on each path between v_i, v_N there is a vertex $w^{\lfloor \frac{r}{2} \rfloor}_{i,N}$ or $\hat{w}^{\lfloor \frac{r}{2} \rfloor}_{i,N}$ at distance exactly r from both v_i, v_N, any selection of a single vertex from inside these two paths will be insufficient to cover both $w^{\lfloor \frac{r}{2} \rfloor}_{i,N}$ and $\hat{w}^{\lfloor \frac{r}{2} \rfloor}_{i,N}$, while selecting either v_i or v_N indeed covers all vertices. □

Block gadget $\hat{G}$: This gadget is the main building block of our construction and uses the above gadgets as inner components. We first make p pairs of paths $A_1, B_1, \ldots, A_p, B_p$ consisting of $2r+1$ vertices each, named $a^0_i, \ldots, a^{2r}_i$ and $b^0_i, \ldots, b^{2r}_i$ for every pair A_i, B_i with $i \in [1, p]$. We then make two copies of the guard gadget $\hat{T}_N$ for A_i, where the $N = 2r+1$ inputs are the vertices $a_0, \ldots, a_{2r}$ and repeat the same for B_i. Figure 3.5 provides an illustration. We refer to all vertices in these gadgets $\hat{T}_{2r+1}$ as the *guards*.

Canonical pairs: We next define $3r+1$ *canonical pairs* of numbers $(\alpha_y \in [0, 2r], \beta_y \in [0, 2r])$, indexed and ordered by $y \in [1, 3r+1]$: for $1 \leq y \leq 2r$, the pair is given by $(\alpha_y = \lfloor \frac{y}{2} \rfloor, \beta_y = \lfloor \frac{y}{2} \rfloor)$ if y is odd, while if y is even the pair is given by $(\alpha_y = y/2 - 1, \beta_y = 2r - y/2 + 1)$. For $2r+1 \leq y \leq 3r+1$ the pair is given by $(\alpha_y = y - r - 1, \beta_y = y - r - 1)$. As an example, the pairs in the correct order for $r = 2$ are (0,0),(0,4),(1,1),(1,3),(2,2),(3,3),(4,4).

In our construction, these pairs will correspond to the indices of the vertices of paths A_i (α_y) and B_i (β_y) that a (canonical) minimum-sized center-set can select. As already mentioned, in the final construction there will be a number of consecutive such pairs of paths A^j_i, B^j_i connected in a path-like manner. We first provide a substitution lemma showing that any minimum-sized center-set does not need to make a selection that does

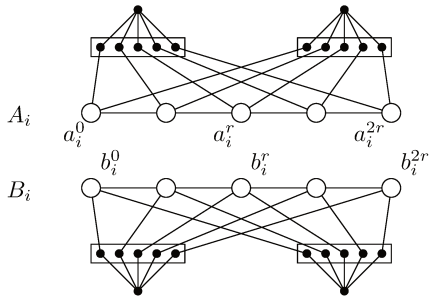


Figure 3.5: The paths A_i, B_i along with their attached guard gadgets for $r = 2$. Note the boxes indicating cliques (for even r).

not correspond to one of the canonical pairs followed by a lemma showing that any center-set only making selections based on canonical pairs will have to respect the ordering of the pairs it uses, i.e. if some pair with index y is used for selection from paths A_i^j, B_i^j, then any pair used in the following paths A_i^{j+1}, B_i^{j+1} must be of index $y' \leq y$. This will form the basis of a crucial argument for showing that any solution will eventually settle into a specific pattern that encodes an assignment without alternations.[1]

Lemma 16. *In a series of M pairs of paths $A_i^j, B_i^j, j \in [1, M]$, where the last vertices of A_i^j, B_i^j are joined with the first vertices of $A_i^{j+1}, B_i^{j+1}, j \in [1, M-1]$, any center-set K of size $|K| = 2m$ that contains one vertex from each path A_i^j, B_i^j can be substituted by a center-set K' of size $|K'| = |K| = 2m$, where the indices of each pair of selected vertices is canonical.*

Proof. First, observe that due to the connection of both final vertices from each pair of paths A_i^j, B_i^j with both first vertices of the following pair A_i^{j+1}, B_i^{j+1}, any selection $a^l \in A_i^j$ and $b^o \in B_i^j$ is equivalent in terms of the vertices it covers with the opposite selection $a^o \in A_i^j$ and $b^l \in B_i^j$, i.e. the same vertices from preceding/succeeding pairs of paths $A_i^{j-1}, B_i^{j-1}/A_i^{j+1}, B_i^{j+1}$ are covered by both selections, while within A_i^j, B_i^j the vertices covered are the opposite. Due to this symmetry, any center-set K in which the index $l \in [0, 2r]$ of the selected vertex $a^l \in A_i^j$ is larger than the index $o \in [0, 2r]$ of the selected vertex $b^o \in B_i^j$, for any $j \in [1, M]$, can be substituted by a center-set K' of the same size in which $l \leq o$ (by replacing one with the other), without affecting the outcome.

Given such a center-set, we claim that any pair of selections $a_j^l \in A_i^j, b_j^o \in B_i^j$ can in fact be substituted by a canonical pair, by showing that *all* pairs can be partitioned in equivalence classes, each being represented by a canonical pair: the center-set can alternate between any of the pairs within each class without any change in vertex coverage and we can thus replace any pair of selections by the canonical pair representing the class to which it belongs.

Consider the canonical pairs for $y \in [1, 2r]$ and odd: the pair is given by $(\alpha_y = \lfloor \frac{y}{2} \rfloor, \beta_y = \lfloor \frac{y}{2} \rfloor)$ and we define the corresponding class of non-canonical pairs to include all pairs where $(\alpha_y = \lfloor \frac{y}{2} \rfloor, \lfloor \frac{y}{2} \rfloor \leq \beta_y \leq 2r - \lceil \frac{y}{2} \rceil)$, i.e. any pair where α_y is the same as the

[1] In fact, the subsequent proof of the converse direction (from (k, r)-center to assignment, Lemma 19) does not require the substitution of Lemma 16 given here, as the structure of the graph itself can enforce all choices made to conform to the above canonical pairs. We offer the substitution lemma as further confirmation of the correctness of our approach.

canonical representative, yet β_y can now range from the same value $\lfloor \frac{y}{2} \rfloor$ up to $2r - \lceil \frac{y}{2} \rceil$. To see why any selections within the class are interchangeable, consider the two extreme cases: let $a_j^{\lfloor \frac{y}{2} \rfloor} \in A_i^j, b_j^{\lfloor \frac{y}{2} \rfloor} \in B_i^j$ be a selection followed by $a_{j+1}^{\lfloor \frac{y}{2} \rfloor} \in A_i^{j+1}, b_{j+1}^{2r-\lceil \frac{y}{2} \rceil}$ in the subsequent pair of paths. Both selections $a_j^{\lfloor \frac{y}{2} \rfloor}, b_j^{\lfloor \frac{y}{2} \rfloor}$ will be at distance $0 < d = r - \lfloor \frac{y}{2} \rfloor \leq r$ from the middle vertices a_j^r, b_j^r on their paths A_i^j, B_i^j and all vertices $a_j^{\lfloor \frac{y}{2} \rfloor + 1}, b_j^{\lfloor \frac{y}{2} \rfloor + 1}, \ldots, a_j^{2r-d}, b_j^{2r-d}$ from the same paths will be covered by these selections. As the selection from A_i^j always matches the one from A_i^{j+1}, the remaining vertices on these paths will be covered by the subsequent selection $a_{j+1}^{\lfloor \frac{y}{2} \rfloor} \in A_i^{j+1}$, as well as all vertices on path B_i^{j+1} up to position $r - \lfloor \frac{y}{2} \rfloor - 2$, meaning the selection from this path can be up to distance $r + 1$ from this "last" covered vertex, giving the index of the furthest possible choice from B_i^{j+1} as $2r - \lfloor \frac{y}{2} \rfloor - 1 = 2r - \lceil \frac{y}{2} \rceil$, being exactly the extremal case for this class. Observe also that this selection will not cover more vertices of the subsequent paths A_i^{j+2}, B_i^{j+2} as it can reach up to vertices at position $r - \lfloor \frac{y}{2} \rfloor - 2$ in both these paths, which are exactly already covered by the selection of $a_{j+2}^{\lfloor \frac{y}{2} \rfloor} \in A_i^{j+2}$, meaning any other intermediate selections would indeed produce the same result as well.

Next, consider the canonical pairs for $y \in [2r + 2, 3r + 1]$: the pair is given by $(\alpha_y = y - r - 1, \beta_y = y - r - 1)$ and we define the corresponding class of non-canonical pairs to include all pairs where $(3r + 2 - y \leq \alpha_y \leq y - r - 1, \beta_y = y - r - 1)$, i.e. any pair where β_y is the same as the canonical representative, yet now α_y can range from the same value $y - r - 1$ down to $3r + 2 - y$. To see why any selections within the class are interchangeable, consider the two extreme cases, as before: let $a_j^{y-r-1} \in A_i^j, b_j^{y-r-1} \in B_i^j$ be a selection followed by $a_{j+1}^{3r+2-y} \in A_i^{j+1}, b_{j+1}^{y-r-1} \in B_i^{j+1}$ in the subsequent pair of paths. Both selections a_j^{y-r-1}, b_j^{y-r-1} will be at distance $0 \leq d = 3r + 1 - y \leq r$ from the final vertices a_j^{2r}, b_j^{2r} on their paths A_i^j, B_i^j and all vertices $a_{j+1}^0, b_{j+1}^0, \ldots, a_{j+1}^{r-d-1}, b_{j+1}^{r-d-1}$ from the following paths A_i^{j+1}, B_i^{j+1} will be covered by these selections. As the selection from B_i^j always matches the one from B_i^{j+1}, all the remaining vertices of B_i^{j+1} are covered, as well as vertices $a_{j+1}^{d'}, \ldots, a_{j+1}^{2r}$ from A_i^{j+1}, where $d' = (y - r - 1) - r + 2d + 2 = 4r + 3 - y$ is the maximum distance in A_i^{j+1} that selection b_{j+1}^{y-r-1} can reach, meaning the selection from this path A_i^{j+1} can be up to distance $r+1$ from this "first" covered vertex (a_{j+1}^{4r+3-y}), giving the index of the nearest possible choice from A_i^{j+1} as $4r+3-y-(r+1) = 3r+2-y$, being exactly the extremal case for this class.

The final observation required for the claim to be shown is that indeed all possible pairs $(l, o) \in [0, 2r]^2$, where $l \leq o$, either exactly match some canonical pair, or are contained in one of the classes given above. As any opposite selections are symmetrical and within each class the actual selections are interchangeable, any arbitrary center-set K can be substituted by a center-set K' of the same size, where all indices of each pair of selections is canonical. □

Lemma 17. *In a series of M pairs of paths $A_i^j, B_i^j, j \in [1, M]$, where the last vertices of A_i^j, B_i^j are joined with the first vertices of A_i^{j+1}, B_i^{j+1} for $j \in [1, M-1]$, in any center-set K that only selects vertices whose indices correspond to canonical pairs, the index y of any canonical pair selected in some pair of paths A_i^j, B_i^j must be larger than, or equal to the index y' of any canonical pair selected in its following pair of paths A_i^{j+1}, B_i^{j+1}.*

Proof. Consider two consecutive pairs of paths A_i^j, B_i^j and A_i^{j+1}, B_i^{j+1} and let $a_j^{\alpha_y}, b_j^{\beta_y}$ and $a_{j+1}^{\alpha_{y'}}, b_{j+1}^{\beta_{y'}}$ be the selections from A_i^j, B_i^j and A_i^{j+1}, B_i^{j+1}, respectively, with $y, y' \in [1, 3r+1]$.

First, for any pair $(\alpha_y = y - r - 1, \beta_y = y - r - 1)$, where $y \in [2r+1, 3r+1]$, both selections $a_j^{\alpha_y}, b_j^{\beta_y}$ will be at distance $0 \leq d = 3r + 1 - y \leq r$ from the final vertices a_j^{2r}, b_j^{2r} on their paths A_i^j, B_i^j and all vertices $a_{j+1}^0, b_{j+1}^0, \ldots, a_{j+1}^{r-d-1}, b_{j+1}^{r-d-1}$ from the following paths A_i^{j+1}, B_i^{j+1} will be covered by these selections. Thus for both paths A_i^{j+1}, B_i^{j+1}, the corresponding distance d' of the selections $a_{j+1}^{\alpha_{y'}}, b_{j+1}^{\beta_{y'}}$ from the final vertices $a_{j+1}^{2r}, b_{j+1}^{2r}$ will be the same for both and at least equal to d, which in turn implies both $\alpha_{y'} \leq \alpha_y$ and $\beta_{y'} \leq \beta_y$, that gives $y' \leq y$ (note that y' can be within $[1, 2r]$ as long as both inequalities hold).

Next, for any pair $(\alpha_y = \lfloor \frac{y}{2} \rfloor, \beta_y = \lfloor \frac{y}{2} \rfloor)$, with $y \in [1, 2r]$ and odd, both selections $a_j^{\alpha_y}, b_j^{\beta_y}$ will be at distance $0 < d = r - \lfloor \frac{y}{2} \rfloor \leq r$ from the middle vertices a_j^r, b_j^r on their paths A_i^j, B_i^j and all vertices $a_j^{\alpha_y+1}, b_j^{\beta_y+1}, \ldots, a_j^{2r-d}, b_j^{2r-d}$ from the same paths will be covered by these selections. Thus in at least one of the following paths A_i^{j+1}, B_i^{j+1} there must be some selection $a_{j+1}^{\alpha_{y'}}$ or $b_{j+1}^{\beta_{y'}}$ at distance d', that is at least equal to d, from either a_{j+1}^r or b_{j+1}^r. As for all pairs (α_y, β_y) it is always $\beta_y \geq \alpha_y$, if this selection is from B_i^{j+1}, then it is $\beta_{y'} \leq \beta_y$ and $\alpha_{y'} \leq \alpha_y$ which gives $y' \leq y$, while if this selection is from A_i^{j+1}, we have $\alpha_{y'} \leq \alpha_y$, which means either also $\beta_{y'} \leq \beta_y$ and thus $y' \leq y$, or $\beta_{y'} = 2r - y'/2 + 1$ and $y' = y + 1$. In this case, observe that $b_{j+1}^{\beta_{y'}}$ covers all vertices $b_{j+1}^{r-y'/2+1}, \ldots, b_{j+1}^{2r-y'/2}$, while $a_{j+1}^{\alpha_{y'}}$ can cover all vertices from b_{j+1}^0 (for $d' > 1$) up to $b_{j+1}^{r-y'/2-1}$ from B_i^{j+1}, thus leaving vertex $b_{j+1}^{r-y'/2}$ at distance $> r$ from any selected vertex.

Finally, for even $y \in [1, 2r]$ and any pair $(\alpha_y = y/2 - 1, \beta_y = 2r - y/2 + 1)$, selected vertex $a_j^{\alpha_y}$ covers all vertices $a_j^{\alpha_y+1}, \ldots, a_j^{\alpha_y+r}$, while selected vertex $b_j^{\beta_y}$ can only cover all vertices from $a_j^{\alpha_y+r+2}$ to a_j^{2r} (for $y < 2r$), thus requiring at least one selection from the following pair of paths A_i^{j+1}, B_i^{j+1}, at distance at most $y/2 - 1$ from the first vertex on its path a_{j+1}^0 or b_{j+1}^0. In either case, we have $\alpha_{y'} \leq \alpha_y$ and thus also $y' \leq y$. □

Now we can continue the description of the block gadget $\hat{G}$. For each pair of paths A_i, B_i with $i \in [1, p]$ we make $3r + 1$ vertices u_i^y, for $y \in [1, 3r + 1]$, that we connect to vertices $a_i^{\alpha_y}$ and $b_i^{\beta_y}$ by paths of length $r+1$, i.e. each vertex u_i^y is at distance $r+1$ from the vertices in A_i, B_i, whose indices match the numbers in the pair corresponding to its own index y (via one path for one such vertex in A_i or B_i). Let the r intermediate vertices on the path from each u_i^y to some vertex in A_i be called $v_i^{y,1} \ldots, v_i^{y,r}$ and the r intermediate vertices on the other path to some vertex in B_i be called $v_i^{y,r+1} \ldots, v_i^{y,2r}$.

Next, we add another vertex q_i that we attach to all vertices u_i^y by paths of length $r-1$ (making the distances between them and q_i equal to r) and then we also attach $3r+1$ paths of length r to q_i, naming the $(2r-1)(3r+1)$ vertices on these paths $q_i^1, \ldots, q_i^{(2r-1)(3r+1)}$. Let U_i be the set of all u_i^y vertices for all $y \in [1, 3r + 1]$ and U be the union of all U_i for $i \in [1, p]$. We then make use of the assignment gadget $\hat{U}_N$ described above by making a copy of $\hat{U}_N$ for each U_i, where the $N = 3r + 1$ inputs are identified with the vertices u_i^y.

Then, for every set $S \subset U$ that contains exactly one vertex from each U_i (the number of such sets being $(3r+1)^p$) we make a vertex x_S. Here we make use of the clique gadget $\hat{X}_N$ described above, where these x_S vertices act as inputs and $N = (3r+1)^p$. Let X be the set

containing all vertices in the gadget, including all x_S vertices. Then, for every vertex x_S we make a copy of the guard gadget $\hat{T}_N$, where the $N = p + 1$ inputs are x_S and the vertices in U for which $u_i^y \in S$ (one from each U_i). This concludes the construction of the block gadget $\hat{G}$.

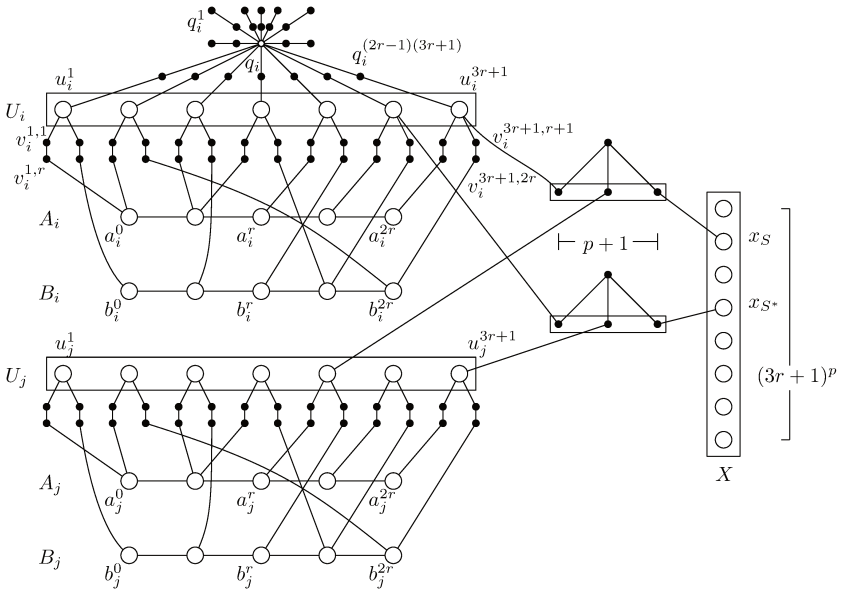


Figure 3.6: Inside a block gadget $\hat{G}$: two pairs of paths A_i, B_i, A_j, B_j with attached U_i, U_j, clique gadget X and guard gadgets between U and X, while guard vertices and all q_j are omitted for clarity. Note boxes around vertices in U_i, U_j, X indicate a gadget, while boxes around the $p + 1$ vertices in the guard gadgets indicate cliques (for even r).

Global construction: Graph G is then constructed as follows. For every group F_τ of variables of ϕ with $\tau \in [1, t]$, we make $m(3rpt+1)$ copies of the gadget $\hat{G}$ that we call $\hat{G}_\tau^\mu$ for $1 \leq \mu \leq m(3rpt + 1)$. Then, for each $\tau \in [1, t]$ we connect gadgets $\hat{G}_\tau^1, \hat{G}_\tau^2, \ldots, \hat{G}_\tau^{m(3rpt+1)}$ in a path-like manner: for every $1 \leq \mu < m(3rpt + 1)$ and $1 \leq i \leq p$, we connect both vertices a_i^{2r}, b_i^{2r} in $\hat{G}_\tau^\mu$ to both vertices a_i^0, b_i^0 in $\hat{G}_\tau^{\mu+1}$. We then make another vertex h that we connect to all vertices a_i^1, b_i^1 in $\hat{G}_\tau^1$ for all $i \in [1, p]$ and $\tau \in [1, t]$, as well as to all vertices a_i^{2r}, b_i^{2r} in $\hat{G}_\tau^{m(3rpt+1)}$ for all $i \in [1, p]$ and $\tau \in [1, t]$, i.e. to all the first and last vertices on the long paths created upon connecting gadgets $\hat{G}_\tau^1, \hat{G}_\tau^2, \ldots, \hat{G}_\tau^{m(3rpt+1)}$. We also attach a path of length r to vertex h, the final vertex of this path named h'.

Next, for every $1 \leq \tau \leq t$ we associate a set $S \subset U$ that contains exactly one vertex from each U_i with an assignment to the variables in group F_τ and as there are at most $2^\gamma = 2^{\lfloor \log_2 (3r+1)^p \rfloor}$ assignments to the variables in F_τ and $(3r + 1)^p \geq 2^\gamma$ sets S, the association can be unique for each τ. For each clause C_π of ϕ with $\pi \in [1, m]$, we make $3rpt + 1$ vertices $\hat{c}_\pi^o$ for $0 \leq o < 3rpt + 1$. Then, to each $\hat{c}_\pi^o$ we attach a path of length $r - 1$ and we consider every assignment to the variables of group F_τ for every $\tau \in [1, t]$ that satisfies the clause C_π: a vertex x_S in $\hat{G}_\tau^{mo+\pi}$, for every $0 \leq o < 3rpt + 1$, is adjacent to the endpoint of this path (thus being at distance r from $\hat{c}_\pi^o$), where S is the subset associated with this assignment within the gadget. This concludes our construction, while Figure 3.7 provides an illustration.

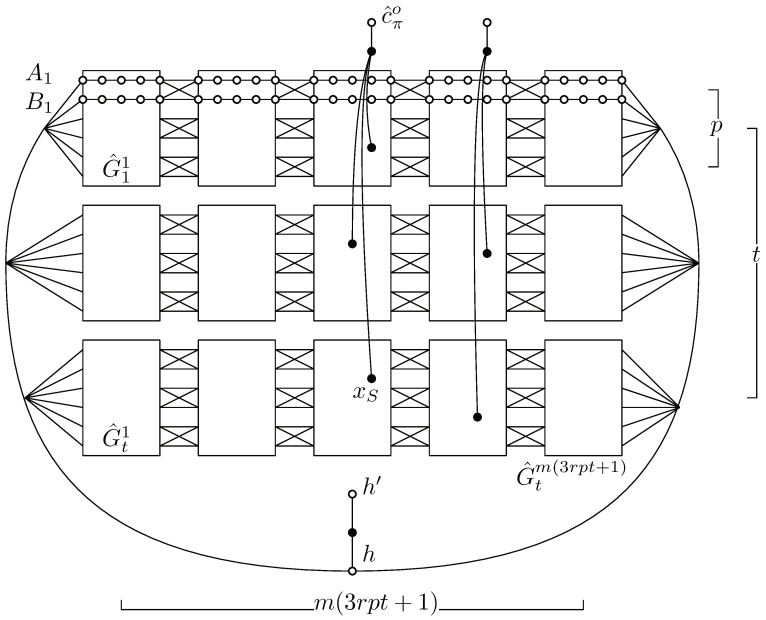


Figure 3.7: A simplified picture of the complete construction. Note boxes indicate block gadgets $\hat{G}$, while there is no vertex anywhere between h and the first/last vertices of the long paths.

Lemma 18. *If ϕ has a satisfying assignment, then G has a (k, r)-center of size $k = ((3r+3)p+1)m(3rpt+1)t+1$.*

Proof. Given a satisfying assignment for ϕ we show the existence of a (k, r)-center K of G of size $|K| = k = ((3r+3)p+1)m(3rpt+1)t+1$. Set K will include the vertex h and $(3r+3)p+1$ vertices from each gadget $\hat{G}^\mu_\tau$, for $\tau \in [1, t]$ and $\mu \in [1, m(3rpt+1)]$. For each group F_τ of variables we consider the restriction of the assignment for ϕ to these variables and identify the set S associated with this restricted assignment. We first add vertex x_S to K and then, for every $i \in [1, p]$, we also add vertex q_i and all $u^z_i \in U_i \setminus S$, as well as the two vertices $a^{\alpha_y}_i$ and $b^{\beta_y}_i$ from the paths A_i, B_i that are connected to the only $u^y_i \in S$ for this i by paths of length $r+1$, i.e. the vertices whose indices $\alpha_y, \beta_y \in [0, 2r]$ correspond to the pair (α_y, β_y) for the index $y \in [1, 3r+1]$ of this vertex. In total, we have $3r$ vertices u^y_i, plus the three $a^{\alpha_y}_i$, $b^{\beta_y}_i$ and q_i for each $i \in [1, p]$, with the addition of vertex x_S completing the selection within each $\hat{G}^\mu_\tau$. Repeating the above for all $m(3rpt+1)t$ gadgets completes the selection for G and what remains is to show that K is indeed a (k, r)-center of G.

First, our selection of one vertex from each path A_i, B_i ensures that all guard vertices are within distance r from some selected vertex. Next, for each $\tau \in [1, t]$ and $i \in [1, p]$, consider the "long paths" formed by joining both vertices a^{2r}_i, b^{2r}_i in $\hat{G}^\mu_\tau$ to both a^0_i, b^0_i in $\hat{G}^{\mu+1}_\tau$. Since the associations between sets S and partial assignments to variables of F_τ are consistent for each τ, the patterns of selection of vertices from each A_i and B_i are repeating, i.e. K contains every $(2r+1)$-th vertex on each path A_i and B_i, meaning all vertices on these paths are within distance r from some selected vertex, apart from the first/last $r-1$ depending on the actual selection pattern, yet these vertices are within distance r from selected vertex h.

Next, within each gadget $\hat{G}^\mu_\tau$, our selection of vertices q_i for every $i \in [1, p]$ brings all

$q_i^1, \ldots, q_i^{(2r-1)(3r+1)}$ and u_i^j vertices within distance r from K. Further, our selection of $a_i^{\alpha_y}, b_i^{\beta_y}$ covers vertices $v_i^{y,1} \ldots, v_i^{y,2r}$ on the two paths from $a_i^{\alpha_y}, b_i^{\beta_y}$ to $u_i^y \in S$, while all other $v_i^{z,w}$ vertices are covered by our selection of each $u_i^z \in U_i \setminus S$. This selection of all $u_i^z \in U_i \setminus S$ also covers all vertices in the assignment gadgets as well as the guard gadgets $\hat{T}_p$ between the u_i^z and $x_{S'}$ for all $S' \neq S$, while selection of x_S covers all vertices in X and all vertices in the guard gadget where x_S is an input.

Finally, concerning the clause vertices $\hat{c}_\pi^o$ for $\pi \in [1, m]$ and $o \in [0, 3rpt]$, observe that if the given assignment for ϕ satisfies the clause C_π, there will be some literal contained therein that is set to true, that corresponds to a variable in a group F_τ for some $\tau \in [1, t]$ and a matching partial assignment for the variables in F_τ associated with some set S^*. Our set K contains vertex x_{S^*} in $\hat{G}_\tau^{mo+\pi}$ for every $o \in [0, 3rpt]$ and vertices $\hat{c}_\pi^o$ are within distance r from x_{S^*}, through a path whose vertices are also covered by x_{S^*}. □

Lemma 19. *If G has a (k, r)-center of size $k = ((3r + 3)p + 1)m(3rpt + 1)t + 1$, then ϕ has a satisfying assignment.*

Proof. Given a (k, r)-center K of G of size $|K| = k = ((3r + 3)p + 1)m(3rpt + 1)t + 1$, we show the existence of a satisfying assignment for ϕ. First, observe that K must contain vertex h, as all vertices on the path attached to it and h' must be within distance r from K. Next we require an averaging argument: as the remaining number of vertices in K is $((3r + 3)p + 1)m(3rpt + 1)t$ and there are $m(3rpt + 1)t$ gadgets $\hat{G}_\tau^\mu$, if there are more than $(3r + 3)p + 1$ vertices selected from some block gadget, then there will be less than these selected from some other block gadget. We will show that not all vertices within a block gadget can be covered by less than $(3r+3)p+1$ selected vertices, which implies that exactly this number is selected from each block gadget in any center-set of size k.

Consider a gadget $\hat{G}_\tau^\mu$. First, observe that at least one of the x_S vertices must be selected to cover all vertices in X and that any such selection indeed covers all vertices in X, as well as all vertices in the guard gadget of which it is an input. This leaves $(3r+3)p$ vertices to cover all other p groups of vertices in the gadget. Observe also that for every $i \in [1, p]$, any minimum-sized center-set must contain one vertex from each path A_i, B_i to cover all the guards, as well as q_i to cover $q_i^1, \ldots, q_i^{(2r-1)(3r+1)}$, as any single selection of some guard vertex will not be sufficient to cover all other guard vertices attached to the same path. This leaves $3rp$ vertices to cover the $v_i^{y,w}$ vertices on the paths between the a_i^j, b_i^l that have not been selected from each pair A_i, B_i, as well as all vertices in the guard gadgets in which each x_{S^*} that was not selected is an input. Due to the structure of the assignment gadgets $\hat{U}$, there must be $3r$ vertices selected from each U_i, thus completing the set K (and the averaging argument). We then claim that the selections from the paths A_i, B_i must match (complement) these selections from U_i, that in turn must match (also complement) the selection of x_S from X.

First, suppose that for some $i \in [1, p]$ the selections a_i^j, b_i^l with $j, l \in [0, 2r]$ from A_i, B_i do not correspond to some canonical pair for some $y \in [1, 3r + 1]$ with $(\alpha_y = j, \beta_y = l)$.[2] Vertex a_i^j will cover at most $2r$ vertices $v_i^{z,1}, \ldots, v_i^{z,r}$ and $v_i^{z',1}, \ldots, v_i^{z',r}$, for $z \neq z'$ (if index j happens to be included in two pairs and only the first r for inclusion in a single pair), but not vertices $v_i^{z,r+1}$ or $v_i^{z',r+1}$. Similarly, vertex b_i^l will also cover at most $2r$ vertices

[2] In fact, Lemma 16 already shows that any (k, r)-center that does not make selections based only on canonical pairs can always be substituted for a (k, r)-center that does. Nevertheless, we show here that this requirement is also enforced by the structure of the graph, mostly for completeness.

$v_i^{w,r+1}, \ldots, v_i^{w,2r}$ and $v_i^{w',r+1}, \ldots, v_i^{w',2r}$, again for $w \neq w'$, but not vertices $v_i^{w,1}$ or $v_i^{w',1}$. Note that since the two choices do not correspond to some canonical pair all these indices will be different: $z \neq z' \neq w \neq w'$. Now, as there are no more selections from A_i, B_i, all vertices $v_i^{y',1}$ and $v_i^{y',r+1}$ are definitely not covered for $y' \neq z, z', w, w'$ (due to any adjacent selections being at distance at least $r+1$). In short, there is at least one vertex $v_i^{y,1}$ or $v_i^{y,r+1}$ (or both) that is not covered for each $y \in [1, 3r+1]$. Since the number of selections from U_i is $3r$ and no other selection would reach these vertices (e.g. from some other $U_{i'}$, due to the guard gadgets employed anywhere in between sets U and X), there will be at least one such vertex that is not covered, implying the selections from every A_i, B_i must match some canonical pair y and the $3r$ selections from U_i must complement this y. Further, suppose the selection x_S from X, for $S = \{u_1^{y_1}, \ldots, u_p^{y_p}\}$, does not match the $3r$ selections from each U_i, i.e. that for some $i \in [1, p]$, it is $u_i^{y_i} \in K \cap U_i$ and $u_i^z \notin K \cap U_i$ for all $z \neq y_i$. Then for set $S^* = S \setminus \{u_i^{y_i}\} \cup \{u_i^z\}$ we have that $x_{S^*} \notin K$ and also $S^* \not\subset K$. This means all vertices in the guard gadget attached to x_{S^*} are not covered.

Next, we require that there exists at least one $o \in [0, 3rpt]$ for every $\tau \in [1, t]$ for which $K \cap \{\bigcup_{i \in [1,p]} A_i \cup B_i\}$ is the same in all gadgets $\hat{G}_\tau^{mo+\pi}$ with $\pi \in [1, m]$, i.e. that there exists a number of successive copies of the gadget for which the pattern of selection of vertices from the paths A_i, B_i does not change. As noted above, set K must contain two vertices $a_i^{\alpha_y}, b_i^{\beta_y}$ from each A_i and B_i, such that the indices α_y, β_y of these two selections match the pair corresponding to the index y of some u_i^y. Consider the "long paths" consisting of paths A_i, B_i sequentially joined with their followers in the next gadget on the same row. Depending on the starting selection, observe that the pattern can "shift towards the left" a number of times in each pair of paths A_i, B_i, as the first and last $r-1$ vertices will be covered by h. That is, a pattern can be selected on some pair A_i, B_i within some gadget $\hat{G}_\tau^\mu$ and a different pattern can be selected on the pair A_i, B_i following it in gadget $\hat{G}_\tau^{\mu+1}$, without affecting whether all vertices on the long paths are covered. As shown by Lemma 17, this can only happen if the index y' that gives the pair of indices $(\alpha_{y'}, \beta_{y'})$ of the second pattern is smaller than or equal to the index y that gives the pair of indices (α_y, β_y) of the first pattern, or $y' \leq y$.

As there are $3r+1$ different indices y and pairs (α_y, β_y), the "shift to the left" can happen at most $3r$ times for each $i \in [1, p]$, thus at most $3rp$ times for each $\tau \in [1, t]$, or $3rpt$ times over all τ. By the pigeonhole principle, there must thus exist an $o \in [0, 3rpt]$ such that no such shift happens among the gadgets $\hat{G}_\tau^{mo+\pi}$, for all $\tau \in [1, t]$ and $\pi \in [1, m]$.

Our assignment for ϕ is then given by the selections for K in each gadget $\hat{G}_\tau^{mo+1}$ for this o: for every group F_τ we consider the selection of $x_S \in X$ that corresponds to a set $S \subset U$, that in turn is associated with a partial assignment for the variables in F_τ. In this way we get an assignment to all the variables of ϕ. To see why this also satisfies every clause C_π with $\pi \in [1, m]$, consider clause vertex $\hat{c}_\pi^o$: this vertex is at distance r from some selected vertex x_S in some gadget $\hat{G}_\tau^{mo+\pi}$. Since the pattern for selection from paths A_i, B_i remains the same in all gadgets $\hat{G}_\tau^{mo+1}, \ldots, \hat{G}_\tau^{mo+\pi}$, so does the set U and also selection of vertices x_S, giving the same assignment for the variables of F_τ associated with S. □

Lemma 20. *Graph G has clique-width $cw(G) \leq tp + f(r, \epsilon)$, for $f(r, \epsilon) = O(r^p)$.*

Proof. We show how to construct graph G using the clique-width operations introduce, join, relabel and at most $f(r, \epsilon)$ labels. We first introduce vertex h and all vertices on the path of length r from it to h', using one label for each vertex, then consecutively join labels/vertices to form the path. The construction will then proceed in a vertical manner,

successively constructing the gadgets $\hat{G}^{\mu}_{\tau}$ for each $\tau \in [1,t]$, before proceeding to repeat the process for each of the $m(3rpt+1)$ columns.

To construct gadget $\hat{G}^{\mu}_{\tau}$ we do the following: we first introduce all guard vertices using one label for each vertex, subsequently applying the appropriate join operations. We do the same for all vertices $q_i, q_i^1, \ldots, q_i^{(2r-1)(3r+1)}$, as well as vertices u_i^y in U_i and all vertices in gadgets $\hat{U}_{3r+1}$ and $v_i^{y,1} \ldots, v_i^{y,2r}$, along with all vertices in X and the guard gadgets $\hat{T}$ attached to each x_S. We have also introduced the clause vertex $\hat{c}^o_{\pi}$ that corresponds to this column and all vertices on the path attached to it, the endpoint of which we now join with matching vertices x_S (if any, for this F_{τ}). We have thus created all vertices (and appropriate edges) in gadget $\hat{G}^{\mu}_{\tau}$, apart from the paths A_i, B_i for $i \in [1,p]$, using one label per vertex. This accounts for the $f(r,\epsilon)$ labels, where function f is $O(r^p)$.

Now, for $i=1$, we introduce vertices a_1^j, b_1^j in turn for each $j \in [0,2r]$, using one label for each vertex and appropriately join each with its previous in the path (for $j \geq 2$), the endpoints of the guard gadgets, as well as the corresponding vertices $v_1^{y,w}$, completing the construction of paths A_1, B_1. We then relabel the vertex b_1^{2r} with the same label as a_1^{2r} (the last vertices of A_1, B_1) and repeat the process for $i = 2, \ldots, p$. When the above has been carried out for all i, the construction of gadget $\hat{G}^{\mu}_{\tau}$ has been completed and we can relabel all vertices to some "junk" label, apart from the two final vertices a_i^{2r}, b_i^{2r} for each $i \in [1,p]$ (that have the same label) and the endpoint of the path of length $r-1$ attached to clause vertex $\hat{c}^o_{\pi}$. This relabelling with some junk label will enable us to reuse the same labels when constructing the same parts of other gadgets. We then repeat the above for the following gadget in the column, until the column is fully constructed. When the column has thus been constructed, we can also relabel the endpoint of the path from clause vertex $\hat{c}^o_{\pi}$ to the junk label, thus reusing its former label for the endpoint of the path attached to the clause vertex of the subsequent column.

During construction of the following column, the first vertices a_i^0, b_i^0 on each path A_i, B_i will both be joined with the label that includes the last vertices a_i^{2r}, b_i^{2r} of the previous column's corresponding path A'_i, B'_i and after each such join, we again relabel these two vertices with the junk label. The construction proceeds in this way until graph G has been fully constructed. Note that during construction of the first and last columns, the first/last vertices of each path have also been joined with vertex h.

In total, the number of labels used simultaneously by the above procedure are the $f(r,\epsilon)$ labels used each time for repeating constructions (also counting the constant number of "outside" labels for h, the clause vertices and the junk label), plus one label for each pair of paths A_i, B_i (containing the last vertices of these paths) in each of the t rows of the construction. The number of these being tp, the claimed bound follows. □

Theorem 21. *For any fixed $r \geq 1$, if (k,r)-*Center *can be solved in $O^*((3r+1-\epsilon)^{cw(G)})$ time for some $\epsilon > 0$, then* SAT *can be solved in $O^*((2-\delta)^n)$ time for some $\delta > 0$.*

Proof. Assuming the existence of some algorithm of running-time $O^*((3r+1-\epsilon)^{\mathrm{cw}(G)}) = O^*((3r+1)^{\lambda \mathrm{cw}(G)})$ for (k,r)-Center, where $\lambda = \log_{3r+1}(3r+1-\epsilon)$, we construct an instance of (k,r)-Center, given a formula ϕ of SAT, using the above construction and then solve the problem using the $O^*((3r+1-\epsilon)^{\mathrm{cw}(G)})$-time algorithm. Correctness is given

by Lemma 18 and Lemma 19, while Lemma 20 gives the upper bound on the running-time:

$$O^*((3r+1)^{\lambda \mathrm{cw}(G)}) \leq O^*\left((3r+1)^{\lambda(tp+f(r,\epsilon))}\right) \tag{3.1}$$

$$\leq O^*\left((3r+1)^{\lambda p\left\lceil \frac{n}{\lfloor \log_2(3r+1)^p \rfloor}\right\rceil}\right) \tag{3.2}$$

$$\leq O^*\left((3r+1)^{\lambda p\frac{n}{\lfloor \log_2(3r+1)^p \rfloor}+\lambda p}\right) \tag{3.3}$$

$$\leq O^*\left((3r+1)^{\lambda \frac{np}{\lfloor p\log_2(3r+1) \rfloor}}\right) \tag{3.4}$$

$$\leq O^*\left((3r+1)^{\delta' \frac{n}{\log_2(3r+1)}}\right) \tag{3.5}$$

$$\leq O^*(2^{\delta'' n}) = O((2-\delta)^n) \tag{3.6}$$

for some $\delta, \delta', \delta'' < 1$. Observe that in line (3.2) the function $f(r,\epsilon)$ is considered constant, as is λp in line (3.4), while in line (3.5) we used the fact that there always exists a $\delta' < 1$ such that $\lambda \frac{p}{\lfloor p\log_2(3r+1) \rfloor} = \frac{\delta'}{\log_2(3r+1)}$, as we have:

$$\begin{aligned} & p\log_2(3r+1) - 1 < \lfloor p\log_2(3r+1) \rfloor \\ \Leftrightarrow\ & \frac{\lambda p\log_2(3r+1)}{p\log_2(3r+1)-1} > \frac{\lambda p\log_2(3r+1)}{\lfloor p\log_2(3r+1) \rfloor}, \\ \text{from which, by substitution, we get:}\ & \frac{\lambda p\log_2(3r+1)}{p\log_2(3r+1)-1} > \delta', \\ \text{now requiring:}\ & \frac{\lambda p\log_2(3r+1)}{p\log_2(3r+1)-1} \leq 1, \\ \text{or:}\ & p \geq \frac{1}{(1-\lambda)\log_2(3r+1)}, \end{aligned}$$

that is precisely our definition of p. This concludes the proof. □

From the above, we directly get the following corollary (for $r = 1$):

Corollary 22. *If* Dominating Set *can be solved in* $O^*((4-\epsilon)^{cw(G)})$ *time for some* $\epsilon > 0$*, then* SAT *can be solved in* $O^*((2-\delta)^n)$ *time for some* $\delta > 0$*.*

3.1.2 Dynamic Programming algorithm

We now present an $O^*((3r+1)^{\mathrm{cw}})$-time dynamic programming algorithm (DP) for unweighted (k,r)-Center, using a given clique-width expression T_G for G with at most cw labels. Even though the algorithm relies on standard techniques, there are several non-trivial, problem-specific observations that we need to make to reach a DP table size of $(3r+1)^{\mathrm{cw}}$.

Our first step is to re-cast the problem as a *distance-labeling* problem, that is, to formulate the problem as that of deciding for each vertex what is its precise distance to the optimal solution K. This is helpful because it allows us to make the constraints of the problem local, and hence easier to verify: roughly speaking, we say that a vertex is satisfied if it has a neighbor with a smaller distance to K (we give a precise definition below). It is now not hard to design a clique-width based DP algorithm for this version of the problem: for each label l we need to remember two numbers, namely the smallest distance value given to some vertex with label l, and the smallest distance value given to a *currently unsatisfied* vertex with label l, if such a vertex exists.

The above scheme directly leads to an algorithm running in time (roughly) $((r+1)^2)^{\mathrm{cw}}$. In order to decrease the size of this table, we now make the following observation: if a label-set contains a vertex at distance i from K, performing a join operation will satisfy all vertices that expect to be at distance $\geq i+2$ from K, since all vertices of the label-set will now be at distance at most 2. This implies that, in a label-set where the minimum assigned value is i, states where the minimum unsatisfied value is between $i+2$ and r are effectively equivalent. With this observation we can bring down the size of the table to $(4r)^{\mathrm{cw}}$, because (intuitively) there are four cases for the smallest unsatisfied value: $i, i+1, \geq i+2$, and the case where all values are satisfied.

The last trick that we need to achieve the promised running-time departs slightly from the standard DP approach. We will say that a label-set is *live* in a node of the clique-width expression if there are still edges to be added to the graph that will be incident to its vertices. During the execution of the dynamic program, we perform a "fore-tracking" step, by checking the part of the graph that comes higher in the expression to determine if a label-set is live. If it is, we merge the case where the smallest unsatisfied value is $i+2$, with the case where all values are satisfied (since a join operation will eventually be performed). Otherwise, a partial solution that contains unsatisfied vertices in a non-live label-set can safely be discarded. This brings down the size of the DP table to $(3r+1)^{\mathrm{cw}}$, and then we need to use some further techniques to make the total running-time quasi-linear in the size of the table. This involves counting the number of solutions instead of directly computing a solution of minimum size, as well as a non-trivial extension of fast subset convolution from [9] for a $3 \times (r+1)$-sized table (or *state-changes*). See also [94, 15] and Chapter 11 of [35].

Distance labeling: We first require an alternative formulation of the problem, based on the existence of a function dl that assigns numbers $\mathrm{dl}(v) \in [0, r]$ to all vertices $v \in V$.

Let $\mathrm{dl} : V \mapsto [0, r]$ and $\mathrm{dl}^{-1}(i)$ be the set of all vertices with assigned number $i \in [0, r]$ by dl. A function dl is then called *valid*, if for all labels $\mathrm{dl} \in [1, \mathrm{cw}]$ and nodes t of the clique-width expression, at least one of the following conditions holds for all vertices $u \in V_l \cap G_t \setminus \mathrm{dl}^{-1}(0)$:

1) There is a neighbor v of u in G_t with a strictly smaller number: $\exists v \in G_t : (u, v) \in E_t \wedge \mathrm{dl}(v) < \mathrm{dl}(u)$;

2) There is a vertex v in the same label l as u and at distance 2 from it in G_t, while their difference in numbers is at least 2: $\exists v \in G_t \cap V_l \wedge \mathrm{dl}(v) \leq \mathrm{dl}(u) - 2 \wedge d_t(u, v) = 2$;

3) There is a vertex v in the same label l as u in G_t with their difference in numbers at least 2 and some vertex w adjacent to it in the final graph G: $\exists v \in G_t \cap V_l \wedge \mathrm{dl}(v) \leq \mathrm{dl}(u) - 2 \wedge \exists w \in G : (u, w) \notin G_t \wedge (u, w) \in G$.

Note that in condition 3) above, vertex w will also be adjacent to u in G (and thus $d(u, v) = 2$), as both u and v belong to the same label. A vertex $u \in V_l \cap G_t$ with label $l \in [1, \mathrm{cw}]$ is *satisfied* by dl for node t, if $\mathrm{dl}(u) = 0$, or either of the first two conditions 1)2) above holds for u. Let $U_l^t(\mathrm{dl})$ be the set of vertices of label l that are not satisfied by dl for node t and $\mathrm{DL}_r(t)$ be the set of all possible valid functions dl for given r, restricted to the vertices of G_t for node t of clique-width expression T_G, with disjoint sets $\mathrm{dl}^{-1}(0)$. The following lemma shows the equivalence between the two formulations.

Lemma 23. *A graph $G = (V, E)$ admits a (k, r)-center if and only if it admits a valid distance-labeling function $dl : V \to \{0, \ldots, r\}$ with $|dl^{-1}(0)| = k$.*

Proof. To see why a valid function dl with $|\mathrm{dl}^{-1}(0)| = k$ represents a solution to the (k, r)-Center problem consider the following: first, given a (k, r)-center of G, let dl be the function that assigns to each vertex $v \in V$ a number equal to its distance from the closest center, i.e. number 0 to the centers, 1 to their immediate neighbors and so on. This function is valid as for every vertex u there always exists some neighbor v with $\mathrm{dl}(v) < \mathrm{dl}(u)$, being the neighbor that lies on the path between u and its closest center, while also $|\mathrm{dl}^{-1}(0)| = k$. On the other hand, given such a valid function dl, the set $\mathrm{dl}^{-1}(0)$ is indeed a (k, r)-center: we have $|\mathrm{dl}^{-1}(0)| = k$ and first, vertices in $\mathrm{dl}^{-1}(1)$ must have a neighbor in the center-set, while vertices in $\mathrm{dl}^{-1}(2)$ are at distance at most 2 from some center. Then, for $i \in [3, r]$, vertices u in $\mathrm{dl}^{-1}(i)$ either have a neighbor in $\mathrm{dl}^{-1}(j)$ with $j < i$, or are at distance at most 2 from some vertex in $\mathrm{dl}^{-1}(j)$ with $j \leq i - 2$, that by induction must in both cases be at distance at most j from some center, making the distance between u and some vertex in $\mathrm{dl}^{-1}(0)$ at most $i \leq r$. □

Table description: There is a table D_t associated with every node t of the clique-width expression, while each table entry $D_t[\kappa, s_1, \ldots, s_w]$, with $w \leq \mathrm{cw}$, is indexed by a number $\kappa \in [0, k]$ and a w-sized tuple $(s_1, \ldots, s_w)$ of *label-states*, assigning a state $s_l = (v_l, u_l)$ to each label $l \in [1, w]$, where $v_l = \min_{x \in V_l} \mathrm{dl}(x) \in [0, r]$ is the minimum number assigned to any vertex x in label l, while $u_l \in \{0, 1, 2\}$ is the difference between v_l and the minimum number assigned by dl to any vertex $y \in U_l^t(\mathrm{dl})$ that is not satisfied by dl for this node: $u_l = 0$ when $\min_{y \in U_l^t(\mathrm{dl})}(\mathrm{dl}(y) - v_l) = 0$, $u_l = 1$ when $\min_{y \in U_l^t(\mathrm{dl})}(\mathrm{dl}(y) - v_l) = 1$ and $u_l = 2$ when $\min_{y \in U_l^t(\mathrm{dl})}(\mathrm{dl}(y) - v_l) \geq 2$, or when $U_l^t(\mathrm{dl}) = \emptyset$. Note that states $(0, 0)$ and $(r, 1)$ do not signify any valid situation and are therefore not used.

There are thus $3r + 1$ possible states for each label, each being a pair signifying the minimum number assigned to any vertex in the label and whether the difference between this and the minimum number of any vertex in the label that is not yet satisfied by dl is either exactly 0,1, or greater than 1, with absence of unsatisfied vertices also considered in the latter case. For a node t with w involved labels, each table entry $D_t[\kappa, s_1, \ldots, s_w]$ contains the number $|\mathrm{DL}_r(t)|$ of valid functions dl restricted to the vertices of G_t with disjoint sets $\mathrm{dl}^{-1}(0)$ and $|\mathrm{dl}^{-1}(0)| = \kappa$, such that for each label $l \in [1, w]$, its state in the tuple gives the conditions that must be satisfied for this label by any such function dl that is to be counted in the entry's value. In particular, we have $\forall t \in T, D_t[\kappa, s_1, \ldots, s_w] : \{\kappa \in [0, k]\} \times \{(1, 0), \ldots, (r, 0), (0, 1), \ldots, (r - 1, 1), (0, 2), \ldots, (r, 2)\}^w \mapsto \mathbb{N}^0$, where $w \in [1, \mathrm{cw}]$.

The inductive computations of table entries for each type of node follows.

Introduce node: For node t with operation $i(l)$ and $l \in [1, \mathrm{cw}]$, we have:

$$D_t[\kappa, s_l] \coloneqq \begin{cases} 1, & \text{if } v_l = 0, u_l = 2, \kappa = 1; \\ 1, & \text{if } v_l \neq 0, u_l = 0, \kappa = 0; \\ 0, & \text{otherwise.} \end{cases}$$

Join node: For node t with operation $\eta(a, b)$, child node $t-1$ and $a, b \in [1, w]$, let $Q(s'_a = (v'_a, u'_a), s'_b = (v'_b, u'_b)) \coloneqq \{(s_a = (v_a, u_a), s_b = (v_b, u_b))|[v_a = v'_a \wedge v_b = v'_b] \wedge [((u'_a = 2 \wedge v_a + u_a > v_b) \vee (u_a = u'_a < 2 \wedge v_a + u_a \leq v_b)) \wedge ((u'_b = 2 \wedge v_b + u_b > v_a) \vee (u_b = u'_b < 2 \wedge v_b + u_b \leq v_a))]\}$. In words, $Q(s'_a, s'_b)$ is the set of all pairs of label states (s_a, s_b), such that if label a is joined with label b, their new states could be s'_a, s'_b, i.e. all pairs of states where the v values remain the same for both a, b, as no new numbers are introduced within any label by a join operation, yet some vertices may become satisfied through the addition of new edges and thus the u values of their label might change to 2. We then have:

$$D_t[\kappa, s_1, \ldots, s'_a, s'_b, \ldots, s_w] \coloneqq \sum_{(s_a, s_b) \in Q(s'_a, s'_b)} D_{t-1}[\kappa, s_1, \ldots, s_a, s_b, \ldots, s_w].$$

Rename node: For node t with operation $\rho(w+1, w)$ and child node $t-1$ (we assume without loss of generality that the last label is renamed into the one preceding it), let $M(s = (v, u)) \coloneqq \{(s_a = (v_a, u_a), s_b = (v_b, u_b))|[v = \min\{v_a, v_b\}] \wedge [[(u = 0) \wedge ((v_a = v \wedge u_a = 0 \wedge v_b \geq v \wedge u_b \geq 0) \vee (v_b = v \wedge u_b = 0 \wedge v_a \geq v \wedge u_a \geq 0))] \vee [(u = 1) \wedge ((v_a = v \wedge u_a = 1 \wedge v_b = v \wedge u_b \geq 1) \vee (v_a = v \wedge u_a \geq 1 \wedge v_b = v \wedge u_b = 1) \vee (v_a = v \wedge u_a = 1 \wedge v_b > v \wedge u_b \geq 0) \vee (v_b = v \wedge u_b = 1 \wedge v_a > v \wedge u_a \geq 0) \vee (v_a = v \wedge u_a = 2 \wedge v_b = v+1 \wedge u_b = 0) \vee (v_b = v \wedge u_b = 2 \wedge v_a = v+1 \wedge u_a \geq 0))] \vee [((u = 2) \wedge (u_a = u_b = 2) \vee (v < v_a \wedge u_b = 2 \wedge ((v_a - v \geq 2 \wedge u_a = 0) \vee (v_a - v \geq 1 \wedge u_a = 1))) \vee (v < v_b \wedge u_a = 2 \wedge ((v_b - v \geq 2 \wedge u_b = 0) \vee (v_b - v \geq 1 \wedge u_b = 1)))]]\}$. In words, $M(s)$ is the set of all pairs of labels (s_a, s_b) for two labels a, b, such that renaming one into the other could produce state s for the resulting label, i.e. the pairs of states where the resulting v value is the minimum of v_a, v_b, while u comes from any of the appropriate combinations of u_a, u_b for each case. We then have:

$$D_t[\kappa, s_1, \ldots, s'_w] \coloneqq \sum_{(s_w, s_{w+1}) \in M(s'_w)} D_{t-1}[\kappa, s_1, \ldots, s_w, s_{w+1}].$$

Union node: For node t with operation $G_{t-1} \cup G_{t-2}$ and children nodes $t-1, t-2$, we assume (again, without loss of generality) that all labels in $[1, y \leq w]$ are involved in $t-1$ and all labels in $[1, z \leq w]$ are involved in $t-2$, such that for some $i \in [1, w]$, all labels $1 \leq j \leq i$ are involved in both nodes $t-1, t-2$, i.e. labels $1, \ldots, i \leq y, z \leq w$ are common to both preceding nodes. Also observe that for any resulting state s'_j of label j, following application of the union operation on two partial solutions where its states are s_j and $\bar{s}_j$, the pair $(s_j, \bar{s}_j)$ would be included in $M(s'_j)$, similarly to renaming j from one partial solution with state s_j to the same label from the other partial solution with state $\bar{s}_j$. We then have:

$$D_t[\kappa, s'_1, \ldots, s'_i, \ldots, s_w] \coloneqq \sum_{\substack{(s_j, \bar{s}_j) \in M(s'_j), j \in [1,i] \\ \kappa_1 \in [0, \kappa]}} (D_{t-1}[\kappa_1, s_1, \ldots, s_i, \ldots, s_y] \cdot D_{t-2}[\kappa - \kappa_1, \bar{s}_1, \ldots, \bar{s}_i, \ldots, s_z]).$$

State changes: Since the number of entries involved in the above computation of a union node's table can be exponential in the clique-width of the graph (due to the number of possible combinations of state pairs in the sum), in order to efficiently compute the tables of union nodes we will use *state changes*: for a union node t we will first transform the tables D_{t-1}, D_{t-2} of its children into tables D^*_{t-1}, D^*_{t-2} of a new type that employs a different state representation, for which the union operation can be efficiently performed to produce table D^*_t, that we finally will transform back to table D_t, thus progressing with our dynamic programming algorithm.

In particular, each entry of table $D^*_t[\kappa, s_1, \ldots, s_w]$ of a node t for $\kappa \in [0, k]$ and $w \in [1, \mathrm{cw}]$ will be an aggregate of entries from $D_t[\kappa, s_1, \ldots, s_w]$, with its value equal to the sum of the appropriate values of the original table. For label l, its state $s^*_l = (v^*_l, u^*_l)$ in the new state signification for table D^*_t will correspond to all states where $v^*_l \geq v_l + i$ and $u^*_l \geq u_l - i$ in the previous state signification for all (applicable) $i \in \{0, 1, 2\}$. Observe that these correspondences exactly parallel the states described in the definition of set $M(s^*_l)$ given above for the computation of a rename node, i.e. all states that could combine to produce the resulting state s^*.

First, let D'_t be a copy of table D_t. The transformation then works in two stages, of w steps each, label-wise: we first produce the intermediate table D'_t from D_t and then table D^*_t from D'_t. For table D'_t we require that all entries $D'_t[\kappa, s'_1, \ldots, s'_w]$ contain the sum of all entries of D_t where $s'_l = (v'_l \geq v_l, u'_l = u_l)$ and all other label-states and κ are fixed: at step l, we first add the entry where $s_l = (r, 0)$ to the entry where $s_l = (r-1, 0)$ and then the entry with $s_l = (r-2, 0)$ to the previous result and so on until $s_l = (1, 0)$. We then repeat this process for $u = 1$ from $s_l = (r-1, 1)$ until $s_l = (0, 1)$ and then for $u = 2$ from $s_l = (r, 2)$ until $s_l = (0, 2)$. We then proceed to the next label until table D'_t is computed.

For the next stage, we initialize D^*_t as a copy of D'_t and also work on w steps, again label-wise, fixing all other label-states and κ: at step l, we first add the entries where $s_l = (v_l, 1)$ and $s_l = (v_l, 2)$ from D'_t to entries where $s_l = (v_l, 0)$ from D^*_t, for each v_l from $r-1$ to 1, in turn (and only the one with $s_l = (r, 2)$ from D'_t for the one where $s_l = (r, 0)$ from D^*_t). We then add the entries where $s_l = (v_l + 1, 0)$ and $s_l = (v_l, 2)$ from D'_t to entries where $s_l = (v_l, 1)$ from D^*_t, for each v_l from r to 0, in turn. Finally, we add the entries where $s_l = (v_l + 1, 1)$ and the entries where $s_l = (v_l + 2, 0)$ from D'_t to entries where $s_l = (v_l, 2)$ from D^*_t, for each v_l from r to 0, in turn. We then proceed to the next step for the following label until table D^*_t is computed. See Figure 3.8 for an illustration of the relationships between states/entries of these tables.

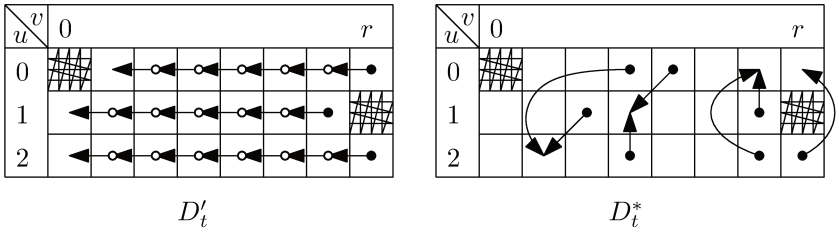


Figure 3.8: A conceptual representation of the relationships between states/entries to be added for the computation of the alternative table types D', D^*. Arrows imply the transfer of values (addition), black endpoints indicate corresponding entries from a previous table type and white endpoints indicate previously computed entries in the same table.

Note that the above procedure is fully reversible,[3] as for each label l, entries where $s_l = (r, 2)$ are the same in all tables D_t, D'_t, D^*_t and thus, to obtain D'_t from D^*_t we again work label-wise, fixing all other label-states and κ: at step l, we first compute the entry of D'_t where $s_l = (r, 0)$, by subtracting from its value the value of the entry where $s_l = (r, 2)$ and we then do the same for the entry where $s_l = (r-1, 2)$, moving to $s_l = (r-1, 1)$ by subtracting the ones where $s_l = (r-1, 2)$ and $s_l = (r, 0)$, then to $s_l = (r-1, 0)$ by subtracting the ones where $s_l = (r-1, 1)$ and $s_l = (r-1, 2)$ and so on, for each v_l from $r-1$ to 0, in turn. After all w steps we have obtained table D'_t and from this we can similarly obtain table D_t, as again, for each label l the entries where $s_l = (r, 0), (r-1, 1), (r, 2)$ are the same: at step l, we compute the entries where $s_l = (v_l, i)$ by subtracting from the corresponding entries of D'_t all entries where $s_l = (v_l + 1, i)$, for v_l from $r-1$ to 0 and $i \in \{0, 1, 2\}$ in turn. For both transformations and directions, we perform at most two additions/subtractions per entry for $\kappa \cdot (3r+1)^{\text{cw}}$ entries, for each step $l \in [1, w \leq \text{cw}]$.

Thus we can compute table D^*_t by simply multiplying the values of the two corresponding entries from D^*_{t-1}, D^*_{t-2}, as they now contain all required information for this state representation, with the inverse transformation of the result giving table D_t:

$$D^*_t[\kappa, s_1, \ldots, s_w] \coloneqq \sum_{\kappa_1=0}^{\kappa_1=\kappa} D^*_{t-1}[\kappa_1, s_1, \ldots, s_w] \cdot D^*_{t-2}[\kappa - \kappa_1, s_1, \ldots, s_w].$$

Fore-tracking: As is already apparent, our algorithm actually solves the counting version of (k, r)-CENTER, by reading the values of entries from table D_z of the final node z, where all labels l are of state $s_l = (v_l, u_l = 2)$. Now, as already noted, these states actually correspond to both situations where either all unsatisfied vertices in the label are of number ≥ 2 than the label's v value, or where there are no unsatisfied vertices in the label for this dl. To ensure our algorithm only counts valid functions dl, we employ a "fore-tracking" policy: whenever some entry is being computed for the table of some rename or union node t, where some label l has state $s_l = (v_l, u_l = 2)$, we establish whether condition 3) given in the definition of a valid function dl is also satisfied by all counted functions dl for all vertices in l, by verifying that some join operation $\eta(l, m)$ is applied between l and another label m in some subsequent node t' of the clique-width expression (even after l is potentially renamed to some other label). If such a join operation is indeed to be applied (and the label-set is live), there will be a vertex $w \in V_m$ that is adjacent to all vertices $u, v \in V_l$ in the final graph G and as $u_l = 2$, there must also be some vertex $v \in V_l$ with $\text{dl}(v) \leq \text{dl}(u) - 2$, for any unsatisfied vertex u. Since it will be $d(u, v) = 2$, all such vertices u will be satisfied in G for all such dl, that will in fact be valid.

On the other hand, in the absence of such a join operation and the case where t is a rename node, we consider the definition of set $M(s)$ above, where there are three options for a resulting state $s_l = (v_l, u_l)$ with $u_l = 2$ from two preceding states $s_a = (v_a, u_a), s_b = (v_b, u_b)$ (corresponding to the last bracket of clauses in the set's definition): we must have either $u_a = u_b = 2$, i.e. that both states have a difference of at least 2 between the minimum number of any vertex and the minimum of any unsatisfied vertex, or $v_l < v_a$ and $u_b = 2$, while either $v_a - v_l \geq 2$ and $u_a = 0$, or $v_a - v_l \geq 1$ and $u_a = 1$ (or vice-versa), i.e. that one label must have state $(> v_l, 0)$ and the other can have any state

[3]This is the reason for *counting* the number of solutions for each κ, instead of finding the *minimum* κ for which at least one solution exists: there is no additive inverse operation for the min-sum semiring, yet the sum-product ring is indeed equipped with subtraction.

from $(\geq v_l + 2, 0)$ or $(\geq v_l + 1, 1)$, their combined numbers giving $(v_l, 2)$. Now, if no join operation follows the rename node t for this label, we simply disregard any options in the above computation (from the last two) that consider states where one of the preceding labels a, b had $u_a, u_b < 2$.

Similarly, for a union node t we consider the state changes given above: if no join node follows t for this label, we simply disregard the additions (and subsequently their corresponding subtractions) for this label in the table's transformation from D'_t to D^*_t in step l, where entries with $s_l = (v_l + 1, 1)$ and $s_l = (v_l + 2, 0)$ from D'_t are added to entries with $s_l = (v_l, 2)$ from D^*_t and keep all such entries as they are (direct copies from D'_t). In this way, as any state where $u = 2$ for some label must be "the direct result" of some join operation (validating condition 1), or there will be some subsequent join operation satisfying all vertices of number ≥ 2 than the minimum (condition 2), no non-valid functions dl for which unsatisfied vertices are infused in the label producing some "false" state where $u = 2$ can be counted, as these vertices are not to be satisfied by some following join node (3).

Correctness: To show correctness of our algorithm we need to establish that for every node $t \in T_G$, each table entry $D_t[\kappa, s_1, \ldots, s_w]$ contains the number of partial solutions to the sub-problem restricted to the graph G_t, i.e. the number of valid functions $\mathrm{dl} \in \mathrm{DL}_r(t)$ with $|\mathrm{dl}^{-1}(0)| = \kappa$ (and disjoint sets $\mathrm{dl}^{-1}(0)$), such that for each label $l \in [1, w]$ (being the labels involved in t) the conditions imposed by its state $s_l = (v_l, u_l)$ are satisfied by all such dl, i.e. $\min_{x \in V_l} \mathrm{dl}(x) = v_l$ and $\min_{y \in U^t_l(\mathrm{dl})}(\mathrm{dl}(y) - v_l) = 0, 1$ when $u_l = 0, 1$, while $\min_{y \in U^t_l(\mathrm{dl})}(\mathrm{dl}(y) - v_l) \geq 2$, or $U^t_l(\mathrm{dl}) = \emptyset$ when $u_l = 2$. That is:

$$\forall t \in T_G, w \in [1, \mathrm{cw}], \forall(\kappa, s_1, \ldots, s_w) \in \tag{3.7}$$

$$\in \{\kappa \in [0, k]\} \times \{(1, 0), \ldots, (r, 0), (0, 1), \ldots, (r - 1, 1), (0, 2), \ldots, (r, 2)\}^w : \tag{3.8}$$

$$\Big\{ D_t[\kappa, s_1, \ldots, s_w] = |\mathrm{DL}_r(t)| : \forall \mathrm{dl} \in \mathrm{DL}_r(t), \forall l \in [1, w] : \tag{3.9}$$

$$\big\{ (\min_{x \in V_l} \mathrm{dl}(x) = v_l) \wedge \tag{3.10}$$

$$\wedge \big[(u_l = 0 \wedge \min_{y \in U^t_l(\mathrm{dl})} (\mathrm{dl}(y) - v_l) = 0) \vee \tag{3.11}$$

$$\vee (u_l = 1 \wedge \min_{y \in U^t_l(\mathrm{dl})} (\mathrm{dl}(y) - v_l) = 1) \vee \tag{3.12}$$

$$\vee (u_l = 2 \wedge \big(\min_{y \in U^t_l(\mathrm{dl})} (\mathrm{dl}(y) - v_l) \geq 2 \vee U^t_l(\mathrm{dl}) = \emptyset \big) \big] \big\} \wedge \tag{3.13}$$

$$\wedge \{ |\mathrm{dl}^{-1}(0)| = \kappa \} \Big\}. \tag{3.14}$$

This is shown by induction on the nodes $t \in T_G$:

- Introduce nodes: This is the base case of our induction. For node t with operation $i(l)$ and $l \in [1, \mathrm{cw}]$, all entries are properly initialized as there is one function dl that includes the introduced vertex in the center-set and one that does not, thus $|\mathrm{DL}_r(t)| = 1$ for $s_l = (0, 2), \kappa = 1$ and $s_l = (> 0, 0), \kappa = 0$, while for any other configuration $\mathrm{DL}_r(t) = \emptyset$. In the following cases, we assume (our induction hypothesis) that all entries of D_{t-1} (and D_{t-2} for union nodes) satisfy the above statement (3.7-3.14).

- Join nodes: For node t with operation $\eta(a, b)$ and $a, b \in [1, w]$, all edges are added between vertices of labels a and b. Thus for each entry $D_t[\kappa, s_1, \ldots, s'_a, s'_b, \ldots, s_w]$, all valid functions dl counted in the entries of the previous table $D_{t-1}[\kappa, s_1, \ldots, s_a, s_b, \ldots, s_w]$ remain valid if $v'_a = v_a, v'_b = v_b$ (3.10) and $u'_a = u_a, u'_b = u_b$ (3.11-3.13), while for all entries in D_t where $v'_a = v_a, v'_b = v_b$ and $u'_a = 2$ (resp. $u'_b = 2$), all functions counted for entries of D_{t-1} where $v_a + u_a > v_b$ (resp. $v_b + u_b > v_a$), are valid as well (3.13), since in the resulting graph G_t of node t, vertices in a (resp. b) can become satisfied by the addition of such edges (and validation of condition 1). This is precisely the definition of set $Q(s'_a, s'_b)$.

- Rename nodes: For node t with operation $\rho(w+1, w)$, all vertices of label $w+1$ will now be included in label w. Thus for each entry $D_t[\kappa, s_1, \ldots, s'_w]$, we require all valid functions dl counted in the entries of the previous table $D_{t-1}[\kappa, s_1, \ldots, s_w, s_{w+1}]$, where states s_w, s_{w+1} describe the possible situations for labels $w, w+1$ that could combine to give the new state s'_w for label w: first, value v'_w denotes the minimum allowed number in any vertex that was previously in w or $w+1$, yet at least one of these labels must have had some vertex with exactly this number and thus $v'_w = \min\{v_w, v_{w+1}\}$ (3.10). Next, value u'_w denotes whether the difference between this number and the minimum number of any unsatisfied vertex is 0,1, or greater (if any) and the possible combinations of states s_w, s_{w+1} are fully described in the definition of set $M(s'_w)$: for u'_w to be 0 we must have $v_w = v'_w$ and $u_w = 0$, while $v_{w+1} \geq v_w$ and $u_{w+1} \geq 0$ (or vice-versa), i.e. at least one of u_w, u_{w+1} must be 0 with its corresponding value v_w, v_{w+1} being the minimum, while the state of the other can have any at least equal values, as only the minimum is retained (3.11). For u'_w to be 1 we must have either $v_w = v'_w$ and $u_w = 1$, while also $v_{w+1} = v'_w$ and $u_{w+1} \geq 1$, or $v_w = v'_w$ and $u_w = 1$, while $v_{w+1} > v'_w$ and $u_{w+1} \geq 0$ (or vice-versa), i.e. one label must have the same target state, while the other must also have the same v with a u greater than 1, or a greater v and any u so as not to make u'_w smaller than the target 1 (3.12). Finally, for u'_w to be 2 we must have either $u_w = u_{w+1} = 2$, i.e. that both states have a difference of at least 2 between the minimum number of any vertex and the minimum of any unsatisfied vertex, or $v'_w < v_w$ and $u_{w+1} = 2$, while either $v_w - v'_w \geq 2$ and $u_w = 0$, or $v_w - v'_w \geq 1$ and $u_w = 1$ (or vice-versa), i.e. that one label must have state $(> v'_w, 0)$ and the other can have any state from $(\geq v'_w + 2, 0)$ or $(\geq v'_w + 1, 1)$, their combined numbers inducing $(v'_w, 2)$ (3.13).

- Union nodes: For node t with operation $G_{t-1} \cup G_{t-2}$ and children nodes $t-1, t-2$, the resulting graph G_t is the disjoint union of graphs G_{t-1}, G_{t-2}, where all common labels $j \in [1, i]$ involved in both $t-1, t-2$ will now include all vertices that were included in label j in one of the previous graphs, while nothing changes for all other labels. This can be seen to be similar to the result of a rename operation between the label j from graph G_{t-1} and the same label from G_{t-2}. Thus for each entry $D_t[\kappa, s'_1, \ldots, s'_i, \ldots, s_w]$, we require the *product* of the numbers of all valid functions dl counted in entries $D_{t-1}[\kappa_1, s_1, \ldots, s_i, \ldots, s_y]$ for $t-1$ and those counted in entries $D_{t-2}[\kappa - \kappa_1, \bar{s}_1, \ldots, \bar{s}_i, \ldots, s_z]$ for $t-2$, summed over all possible combinations of states $s_j, \bar{s}_j$ that belong to some pair in $M(s'_j)$ for every $j \in [1, i]$, as well as over all values of κ_1 from 0 to κ, since the number of valid functions in G_t respecting the conditions of the target entry is equal to a number of valid functions in G_{t-1} (counted in entries of the first type) for each valid function that complements each

of them in G_{t-2} (counted in entries of the second type): the sizes of $\mathrm{dl}^{-1}(0)$ always add up to κ (3.14), while the states $s_j, \bar{s}_j$ of all common labels j in any pair of entries from D_{t-1}, D_{t-2} would result in the desired target state s_j for the same label in the entry from D_t in question, as they belong in the set $M(s'_j)$ (3.10-3.13).

Theorem 24. *Given graph G, along with $k, r \in \mathbb{N}^+$ and clique-width expression T_G of clique-width cw for G, there exists an algorithm to solve the counting version of the (k,r)-*Center *problem in $O^*((3r+1)^{cw})$ time.*

Proof. Correctness of the dynamic programming algorithm is given above, while for the final computation at the root z of T_G, all entries $D_z[k, s_1, \ldots, s_w]$ with $u_l = 2, \forall l \in [1, w]$ and $w \in [1, \mathrm{cw}]$ can be considered for identification of the number of valid functions dl, or (k,r)-centers of graph $G_z = G$. For the decision version of the problem, the algorithm can output "yes" if any of these entries' value is > 0 and "no" otherwise.

For the algorithm's complexity, there are at most $k(3r+1)^{\mathrm{cw}}$ entries in each table D_t of any node t and any entry of the $O(n \cdot \mathrm{cw}^2)$ join/rename nodes can be computed in $O(r)$ time, while for the $O(n)$ union nodes, table transformations require $O^*(k \cdot \mathrm{cw} \cdot (3r+1)^{\mathrm{cw}})$ time and any entry of the transformed tables can be computed in $O(k)$ time. □

3.2 Vertex Cover, Feedback Vertex Set and Tree-depth

3.2.1 Vertex Cover and Feedback Vertex Set: W[1]-hardness

In this subsection we first show that the edge-weighted variant of the (k,r)-Center problem parameterized by $\mathrm{vc} + k$ is W[1]-hard, and more precisely, that the problem does not admit a $n^{o(\mathrm{vc}+k)}$ algorithm under the ETH (Theorem 27). We give a reduction from k-Multicolored Independent Set. This is a well-known W[1]-hard problem that cannot be solved in $n^{o(k)}$ under the ETH [35].

Construction: Given an instance $[G = (V, E), k]$ of k-Multicolored Independent Set, we will construct an instance $[G' = (V', E'), k]$ of edge-weighted (k,r)-Center, where $r = 3n$. First, for every set $V_i \subseteq V$, we create a set $P_i \subseteq V'$ of n vertices $p_l^i, \forall l \in [1, n]$, $\forall i \in [1, k]$ (that directly correspond to the vertices of V_i) and two *guard* vertices g_i^1, g_i^2, attaching them to all vertices in P_i by edges of weight $r = 3n$. Next, for each $i \in [1, k]$, we create another pair of vertices a_i, b_i and connect a_i to each vertex p_l^i by an edge of weight $n + l$, while b_i is connected to each vertex p_l^i by an edge of weight $2n - l + 1$. Now each P_i contains all vertices a_i, b_i, g_i^1, g_i^2 and each $p_l^i, \forall l \in [1, n]$.

Finally, for each edge $e \in E$ with endpoints in V_{i_1}, V_{i_2} and $i_1 \neq i_2$ (not part of a clique), we create a vertex u_e that we connect to vertices a_{i_1}, b_{i_1} and a_{i_2}, b_{i_2}. We set the weights of these edges as follows: suppose that e connects the j_1-th vertex of V_{i_1} to the j_2-th vertex of V_{i_2}. Then we set $w(u_e, a_{i_1}) = 2n - j_1 + 1$, $w(u_e, b_{i_1}) = n + j_1$, $w(u_e, a_{i_2}) = 2n - j_2 + 1$, $w(u_e, b_{i_2}) = n + j_2$.

Lemma 25. *If G has a k-multicolored independent set, then G' has a $(k, 3n)$-center.*

Proof. Let $I \subseteq V$ be a multicolored independent set in G of size k and $v_{l_i}^i$ denote the vertex selected from each V_i, or $I \coloneqq \{v_{l_1}^1, \ldots, v_{l_i}^i, \ldots, v_{l_k}^k\}$. Let $S \subseteq V'$ be the set of vertices $p_{l_i}^i$ in G' that correspond to each $v_{l_i}^i$. We claim S is a (k,r)-center of G': since one vertex is selected from each P_i in G', all the guards g_i^1, g_i^2 and vertices a_i, b_i are within distance

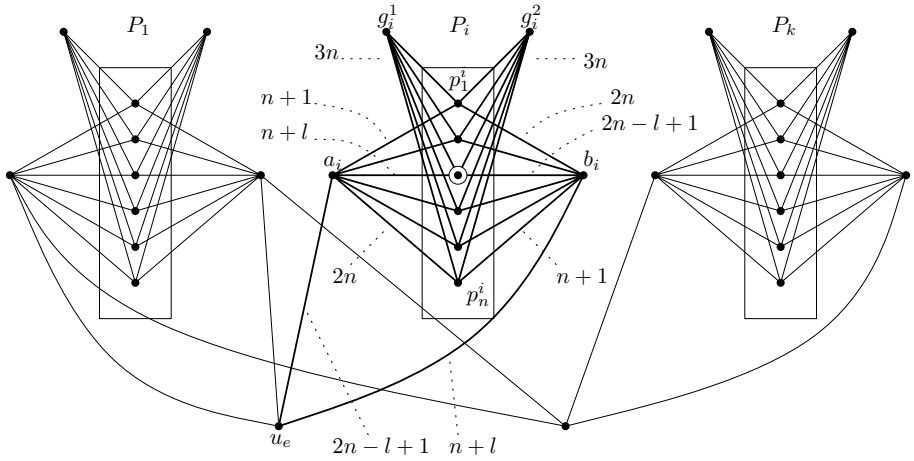


Figure 3.9: A general picture of graph G'. The circled vertex is p_l^i, while dotted lines match edges to weights.

$r = 3n$ from selected vertex $p_{l_i}^i$ for some $l_i \in [1, n]$ and every $i \in [1, k]$. For vertices u_e, observe that selected vertex $p_{l_i}^i$ is at distance $n + l_i + 2n - l_i + 1 = 3n + 1$ from u_e, through either a_i or b_i, if its corresponding vertex in G is an endpoint of e, or $e = (v_{l_i}^i, w)$ for some $w \in V \setminus V_i$. As I is an independent set of G, however, there can be no edge between any two selected vertices $v_{l_i}^i, v_{l_j}^j \in I$ with $i \neq j$ and thus, we know that for every u_e at least one of the vertices $p_{l_i}^i, p_{l_j}^j$ selected for S from the two components P_i, P_j to which u_e is connected will not be one of the vertices corresponding to the endpoints of edge e, or $e \neq (v_{l_i}^i, v_{l_j}^j)$. Let $e := (v_x^i, v_y^j)$, with $x \neq l_i$ and/or $y \neq l_j$. Assuming without loss of generality that $x \neq l_i$ is the case, we have that the distances from u_e to $p_{l_i}^i$ are $n + l_i + 2n - x + 1$ via a_i and $2n - l_i + n + x$ via b_i. If $x > l_i$ then distance via a_i is $3n + 1 + l_i - x \leq 3n$, while if $x < l_i$ the distance via b_i is $3n + x - l_i < 3n$. □

Lemma 26. *If G' has a $(k, 3n)$-center, then G has a k-multicolored independent set.*

Proof. Let $S \subseteq V'$ be the $(k, 3n)$-center in G'. As $|S| = k$ and all guard vertices $g_i^1, g_i^2, \forall i \in [1, k]$ must be within distance $3n$ from some selected vertex, set S must contain exactly one vertex from each P_i, or $S = \{p_{l_1}^1, \ldots, p_{l_k}^k\}$ for some $l_i \in [1, n]$ for each $i \in [1, k]$. We let $I \subseteq V$ be the set of vertices $v_{l_i}^i \in V_i, i \in [1, k]$ that correspond to each $p_{l_i}^i$ and claim that I is an independent set in G: suppose there is an edge $e = (v_{l_i}^i, v_{l_j}^j) \in E, i \neq j$ and $v_{l_i}^i, v_{l_j}^j \in I$. Then there must be a vertex $u_e \in G'$ with edges to $a_i, b_i \in P_i$ and $a_j, b_j \in P_j$, where we have $p_{l_i}^i, p_{l_j}^j \in S$. The distance from u_e to $p_{l_i}^i$ is $2n - l_i + 1 + n + l_i > 3n$, via either a_i or b_i, while the distance from u_e to $p_{l_j}^j$ is also $2n - l_j + 1 + n + l_j > 3n$, via either a_j, b_j, meaning u_e is not covered by S, giving a contradiction. □

Theorem 27. *The weighted (k, r)-Center problem is W[1]-hard parameterized by $vc + k$. Furthermore, if there is an algorithm for weighted (k, r)-Center running in time $n^{o(vc+k)}$ then the ETH is false.*

Proof. Observe that the set $Q \subset V'$ that includes all guard vertices g_i^1, g_i^2 and a_i, b_i, $\forall i \in [1, k]$, is a vertex cover of G', as all edges have exactly one endpoint in Q. This

means $\mathrm{vc}(G') \leq 4k$. In addition, parameter k remains the same in both the instances of k-Multicolored Independent Set and (k,r)-Center. Thus, the construction along with Lemmas 25 and 26, indeed imply the statement. □

Using essentially the same reduction we also obtain a similar hardness result for unweighted (k,r)-Center parameterized by fvs.

Corollary 28. *The (k,r)-Center problem is W[1]-hard when parameterized by fvs $+ k$. Furthermore, if there is an algorithm for weighted (k,r)-Center running in time $n^{o(vc+k)}$, then the ETH is false.*

Proof. We use the same reduction as in Theorem 27, except that we replace all weighted edges by unweighted paths through new vertices in such a way that distances between original vertices are preserved. It is not hard to see that any set that was a vertex cover of the previous graph is a feedback vertex set of the new graph, hence $\mathrm{fvs} = O(k)$. Lemma 25 goes through unchanged, while for Lemma 26 it suffices to observe that, because of the guard vertices, no valid solution can be using one of the new vertices as a center. □

3.2.2 Vertex Cover: FPT algorithm

We now show that unweighted (k,r)-Center admits an algorithm running in time $O^*(5^{\mathrm{vc}})$, in contrast to its weighted version (Theorem 27). We devise an algorithm that operates in two stages: first, it guesses the intersection of the optimal solution with the optimal vertex cover, and then it uses a reduction to Set Cover to complete the solution.

Theorem 29. *Given graph G, along with $k, r \in \mathbb{N}^+$ and a vertex cover of size vc of G, there exists an algorithm solving unweighted (k,r)-Center in $O^*(5^{vc})$ time.*

Proof. Let C be the given vertex cover of G and $I = V \setminus C$ be the remaining independent set. We assume without loss of generality that the graph is connected (otherwise each component can be handled separately). We also assume that $r \geq 2$, otherwise the problem reduces to Dominating Set which is already known to be solvable faster than $O^*(5^{\mathrm{vc}})$.

Let K be some (unknown) optimal solution. Our algorithm first guesses a partition of C into three sets $C = S \cup R \cup Q$ such that $S = K \cap C$, Q contains the vertices of C which are at distance exactly r from K, and R the rest (which are at distance $< r$ from K). Our algorithm first guesses this partition by trying out all 3^{vc} possibilities.

Suppose we are given a partition $C = S \cup R \cup Q$ as above. We would like to check if this is the correct partition and then find a set $Z \subseteq I$ such that $S \cup I$ is an optimal solution. First, we verify that all vertices of Q are at distance $\geq r$ from S (otherwise we already know this is not a correct partition). Second, we check if there exists $v \in I$ such that $N(v) \subseteq Q$. In such a case, we again know that this is not a correct partition, since such a v would need to be included in K, which would imply that its neighbors in Q are at distance $< r$ from K. We can therefore assume that all vertices in I have some neighbor in $S \cup R$.

We now formulate an instance of Set Cover as follows: the universe contains all vertices of $R \cup Q$ which are not already covered, that is, all vertices of R which are at distance $\geq r$ from S and all vertices of Q which are at distance $> r$ from S. We construct a set for each vertex of $v \in I$ and we place in it all vertices $u \in R$ such that $d(v,u) < r$ and all vertices $u \in Q$ such that $d(v,u) \leq r$. We solve this Set Cover instance in time $O^*(2^{|R \cup Q|})$ using dynamic programming, and this gives us a set $Z \subseteq I$. We output $S \cup Z$

as a solution to (k,r)-Center. We observe that this is always a valid solution because by construction all vertices of R are at distance $< r$ from $S \cup Z$, and all vertices of I have a neighbor in $S \cup R$. If we started with the correct partition of C into S, R, Q then this solution is optimal because $K \cap I$ must give a feasible set cover of the instance we constructed.

To analyze the running-time, observe that if $|S| = i$, then the algorithm goes through $2^{\mathrm{vc}-i}$ possible partitions of $C \setminus S$ into R, Q, and then for each partition spends $2^{\mathrm{vc}-i}n^{O(1)}$ to solve Set Cover. Hence, the algorithm runs in time $\sum_{i=0}^{\mathrm{vc}} \binom{vc}{i} 4^{\mathrm{vc}-i} n^{O(1)} = \sum_{i=0}^{\mathrm{vc}} \binom{\mathrm{vc}}{i} 4^i n^{O(1)} = 5^{\mathrm{vc}} n^{O(1)}$. □

3.2.3 Tree-depth: Tight ETH-based lower bound

We now consider the unweighted version of (k,r)-Center parameterized by td. Theorem 27 has established that weighted (k,r)-Center is W[1]-hard parameterized by vc (and hence also by td by Lemma 3), but the complexity of unweighted (k,r)-Center parameterized by td does not follow from this theorem, since td is incomparable to fvs. Indeed, we will show that (k,r)-Center is FPT parameterized by td and precisely determine its parameter dependence based on the ETH.

We begin with a simple upper bound argument. We will make use of the following known fact on tree-depth, while the algorithm then follows from the dynamic programming procedure of [22] and the relationship between r, td and tw:

Lemma 30. *For any graph $G = (V, E)$ we have $diam(G) \leq 2^{td+1} - 2$, where $diam(G)$ denotes the graph's diameter.*

Proof. We use the following equivalent inductive definition of tree-depth: $\mathrm{td}(K_1) = 0$ while for any other graph $G = (V, E)$ we set $\mathrm{td}(G) = 1 + \min_{u \in V} \mathrm{td}(G \setminus u)$ if G is connected, and $\mathrm{td}(G) = \max_C \mathrm{td}(G[C])$ if G is disconnected, where the maximum ranges over all connected components of G.

We prove the claim by induction. The inequality is true for K_1, whose diameter is 0. For the inductive step, the interesting case is when $G = (V, E)$ is connected, since otherwise we can assume that the claim has been shown for each connected component and we are done. Let $u \in V$ be such that $\mathrm{td}(G) = 1 + \mathrm{td}(G \setminus u)$. Consider two vertices $v_1, v_2 \in V \setminus \{u\}$ which are at maximum distance in G. If v_1, v_2 are in the same connected component of $G' := G \setminus u$, then $d_G(v_1, v_2) \leq d_{G'}(v_1, v_2) \leq diam(G') \leq 2^{\mathrm{td}(G')+1} - 2 \leq 2^{\mathrm{td}(G)+1} - 2$, where we have used the inductive hypothesis on G'. So, suppose that v_1, v_2 are in different connected components of G'. It must be the case that u has a neighbor in the component of v_1 (call it v_1') and in the component of v_2 (call it v_2'), because G is connected. We have $d_G(v_1, v_2) \leq d_G(v_1, v_1') + 2 + d_G(v_2, v_2') \leq d_{G'}(v_1, v_1') + 2 + d_{G'}(v_2, v_2') \leq 2diam(G') + 2 \leq 2 \cdot 2^{\mathrm{td}(G')+1} - 2 = 2^{\mathrm{td}(G)+1} - 2$. □

Theorem 31. *Unweighted (k,r)-Center can be solved in time $O^*(2^{O(td^2)})$.*

Proof. The main observation is that we can assume that $r \leq diam(G)$, because otherwise the problem is trivial. Hence, by Lemma 30 we have $r \leq 2^{\mathrm{td}+1}$. We can now rely on Lemma 3 to get $\mathrm{tw} \leq \mathrm{td}$, and the algorithm of [22] which runs in time $O^*((2r+1)^{\mathrm{tw}})$ gives the desired running-time. □

The main contribution concerning the tree-depth parameter is to establish a tight ETH-based lower bound, matching Theorem 31.

Construction: Given an instance ϕ of 3-SAT on n variables and m clauses, where we can assume that $m = O(n)$ (by the *Sparsification Lemma*, see [61]), we will create an instance $[G = (V, E), k]$ of the unweighted (k, r)-CENTER problem, where $k = \sqrt{n}$ and $r = 3 \cdot c^{\sqrt{n}}$ for an appropriate constant c. To simplify notation, we assume without loss of generality that $\sqrt{n}$ is an integer.

We first group the clauses of ϕ into $\sqrt{n}$ equal-sized groups $F_1, \ldots, F_{\sqrt{n}}$. As a result, each group involves $O(\sqrt{n})$ variables, so there are $2^{O(\sqrt{n})}$ possible assignments to the variables of each group. Select c appropriately so that each group F_i has at most $c^{\sqrt{n}}$ possible partial assignments ϕ_j^i for the variables of clauses in F_i.

We then create for each $i \in \{1, \ldots, n\}$, a set P_i of at most $c^{\sqrt{n}}$ vertices $p_1^i, \ldots, p_{c^{\sqrt{n}}}^i$, such that each vertex of P_i represents a partial assignment to the variables of F_i that satisfies all clauses of F_i. We add two *guard* vertices g_i^1, g_i^2, attaching them to all vertices of P_i by paths of length $r = 3 \cdot c^{\sqrt{n}}$. Next, for each $i \in \{1, \ldots, \sqrt{n}\}$, we create another pair of vertices a_i, b_i, connecting a_i to each vertex p_l^i by a path of length $c^{\sqrt{n}} + l$, while b_i is connected to each vertex p_l^i by a path of length $2 \cdot c^{\sqrt{n}} - l + 1$. Now each P_i contains all vertices a_i, b_i, g_i^1, g_i^2 and each $p_l^i, \forall l \in \{1, \ldots, c^{\sqrt{n}}\}$.

Finally, for any two *conflicting* partial assignments ϕ_l^i, ϕ_o^j, with $l, o \in \{1, c^{\sqrt{n}}\}$ and $i, j \in \{1, \sqrt{n}\}$, i.e. two partial assignments that assign conflicting values to at least one variable, we create a vertex $u_{l,o}^{i,j}$ that we connect to vertices a_i, b_i and a_j, b_j: if $p_l^i \in P_i$ is the vertex corresponding to ϕ_l^i and $p_o^j \in P_j$ is the vertex corresponding to ϕ_o^j, then vertex $u_{l,o}^{i,j}$ is connected to a_i by a path of length $2 \cdot c^{\sqrt{n}} - l + 1$ and to b_i by a path of length $c^{\sqrt{n}} + l$, as well as to a_j by a path of length $2 \cdot c^{\sqrt{n}} - o + 1$ and b_j by a path of length $c^{\sqrt{n}} + o$. See Figure 3.10 for an illustration.

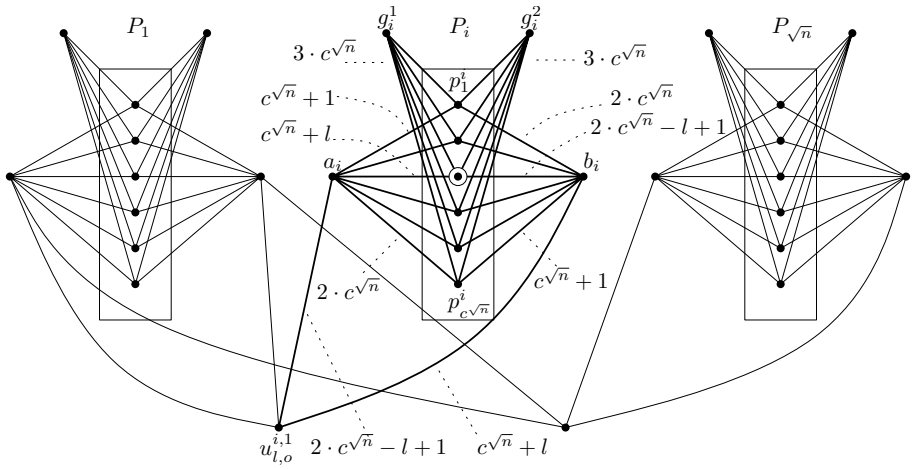


Figure 3.10: A general picture of graph G. Note all straight lines denote paths of length equal to the number shown by dotted lines, while the circled vertex is p_l^i.

Lemma 32. *If ϕ has a satisfying assignment, then there exists a (k, r)-center in G of size $k = \sqrt{n}$ and $r = 3 \cdot c^{\sqrt{n}}$.*

Proof. Consider the satisfying assignment for ϕ and let $\phi_{l_i}^i$, with $l_i \in \{1, \ldots, c^{\sqrt{n}}\}$ and $i \in \{1, \ldots, \sqrt{n}\}$, be the restriction of that assignment for all variables appearing in clauses of group F_i. We claim the set K, consisting of all vertices $p_{l_i}^i$ corresponding to $\phi_{l_i}^i$,

is a (k, r)-center for G with $k = \sqrt{n}$ and $r = 3 \cdot c^{\sqrt{n}}$. Since one vertex is selected from each P_i in G, all the guards g_i^1, g_i^2 and vertices a_i, b_i are within distance $3 \cdot c^{\sqrt{n}}$ from selected vertex $p_{l_i}^i$. For vertices $u_{l_i,l_j}^{i,j}$, observe that selected vertex $p_{l_i}^i$ is at distance $c^{\sqrt{n}} + l_i + 2 \cdot c^{\sqrt{n}} - l_i + 1 = 3 \cdot c^{\sqrt{n}} + 1$ from $u_{l_i,l_j}^{i,j}$, through either a_i or b_i, while vertex $p_{l_j}^j$ is at distance $c^{\sqrt{n}} + l_j + 2 \cdot c^{\sqrt{n}} - l_j + 1 = 3 \cdot c^{\sqrt{n}} + 1$ from $u_{l_i,l_j}^{i,j}$, through either a_j or b_j, only if the corresponding partial assignments $\phi_{l_i}^i$ and $\phi_{l_j}^j$ are conflicting. As $\phi_{l_i}^i$, $\phi_{l_j}^j$ are parts of a satisfying assignment for ϕ, however, this will not be the case and at least one path from $u_{l_i,l_j}^{i,j}$ to either $p_{l_i}^i$, or $p_{l_j}^j$, will be of length $\leq 3 \cdot c^{\sqrt{n}}$. □

Lemma 33. *If there exists a (k, r)-center in G of size $k = \sqrt{n}$ and $r = 3 \cdot c^{\sqrt{n}}$, then ϕ has a satisfying assignment.*

Proof. Let $K \subset V$ be the (k, r)-center in G, with $r = 3 \cdot c^{\sqrt{n}}$ and $k = \sqrt{n}$. As $|K| = k = \sqrt{n}$ and all guard vertices $g_i^1, g_i^2, \forall i \in \{1, \ldots, \sqrt{n}\}$, must be within distance at most $3 \cdot c^{\sqrt{n}}$ from some vertex selected in K, the set K must contain exactly one vertex from each P_i, or $K = \{p_{l_1}^1, \ldots, p_{l_{\sqrt{n}}}^{\sqrt{n}}\}$, for some $l_i \in \{1, \ldots, c^{\sqrt{n}}\}$ and for each $i \in \{1, \ldots, \sqrt{n}\}$. We set the assignment for ϕ to consist of all partial assignments $\phi_{l_i}^i$, with $i \in \{1, \ldots, \sqrt{n}\}$, corresponding to vertices $p_{l_i}^i \in K$ and claim that this is a valid assignment that satisfies ϕ. It is not hard to see that, if the assignment is valid, then it satisfies the clause, as we have only listed partial assignments that satisfy all clauses of each group. To see that the assignment does not assign conflicting values to any variable, suppose for contradiction that there are two conflicting partial assignments $\phi_{l_i}^i, \phi_{l_j}^j$ and a vertex $u_{l_i,l_j}^{i,j} \in G$ with paths to $a_i, b_i \in P_i$ and $a_j, b_j \in P_j$, where we have $p_{l_i}^i, p_{l_j}^j \in K$. The distance from $u_{l_i,l_j}^{i,j}$ to $p_{l_i}^i$ is $2 \cdot c^{\sqrt{n}} - l_i + 1 + c^{\sqrt{n}} + l_i > 3 \cdot c^{\sqrt{n}}$, via either a_i or b_i, while its distance to $p_{l_j}^j$ is also $2 \cdot c^{\sqrt{n}} - l_j + 1 + c^{\sqrt{n}} + l_j > 3 \cdot c^{\sqrt{n}}$, via either a_j, b_j, meaning $u_{l_i,l_j}^{i,j}$ is not covered by K, giving a contradiction. □

Lemma 34. *The tree-depth of G is $4\sqrt{n} + \lceil \log(3 \cdot c^{\sqrt{n}}) \rceil + 1 = O(\sqrt{n})$.*

Proof. Again we use the definition of tree-depth used in Lemma 30. Consider the graph G after removal of all guard vertices g_i^1, g_i^2 and all $a_i, b_i, \forall i \in [1, \sqrt{n}]$. The graph now consists of a number of sub-divided stars, centered either on some vertex $u_{l,o}^{i,j}$, or some p_l^i, while the maximum distance from each such center to a leaf vertex in the star is $3 \cdot c^{\sqrt{n}} - 1$, for a path connecting p_l^i to a guard g_i^1, g_i^2, omitting the final edge due to removal of g_i^1, g_i^2. The claim follows since paths of length n have tree-depth exactly $\lceil \log(n+1) \rceil$ (this can be shown by repeatedly removing the middle vertex of a path). By the definition of tree-depth, after removal of $4\sqrt{n}$ vertices from G, the maximum tree-depth of each resulting disconnected component is $\lceil \log(3 \cdot c^{\sqrt{n}}) \rceil = \lceil \sqrt{n} \cdot \log(c) + \log(3) \rceil$. □

Theorem 35. *If (k, r)-Center can be solved in $2^{o(td^2)} \cdot n^{O(1)}$ time, then 3-SAT can be solved in $2^{o(n)}$ time.*

Proof. Suppose there is an algorithm for the (k, r)-Center problem with running-time $2^{o(\mathrm{td}^2)}$. Given an instance ϕ of 3-SAT, we use the above construction to create an instance $[G, k, r]$ of (k, r)-Center, with $k = \sqrt{n}$ and $r = 3 \cdot c^{\sqrt{n}}$, in time $O(\sqrt{n} \cdot c^{\sqrt{n}} + c^{2\sqrt{n}})$. As, by Lemma 34, we have $\mathrm{td}(G) = O(\sqrt{n})$, using the supposed algorithm for (k, r)-Center, we can decide whether ϕ has a satisfying assignment in time $2^{o(\mathrm{td}^2)} \cdot n^{O(1)} = 2^{o(n)}$. □

3.3 Treewidth: FPT approximation scheme

In this section we present an FPT approximation *scheme* (FPT-AS) for (k, r)-Center parameterized by tw. Given as input a weighted graph $G = (V, E)$, $k, r \in \mathbb{N}^+$ and an arbitrarily small error parameter $\epsilon > 0$, our algorithm is able to return a solution that uses a set of k centers K, such that all other vertices are at distance at most $(1+\epsilon)r$ from K, or to correctly conclude that no (k, r)-center exists. The running-time of the algorithm is $O^*((\text{tw}/\epsilon)^{O(\text{tw})})$, which (for large r) significantly out-performs any *exact* algorithm for the problem (even for the unweighted case and more restricted parameters, as in Theorems 27 and 28), while only sacrificing a small ϵ error in the quality of the solution.

Our algorithm is a modification of the dynamic programming for (k, r)-Center parameterized by tw, that we provide for completeness at the end of this chapter (Theorem 46, see also [22]). Our approach will rely heavily on a technique introduced in [73] (see also [4]) to approximate problems which are W-hard by treewidth. The idea is that, if the hardness of the problem is due to the fact that the DP table needs to store tw large numbers (in our case, the distances of the vertices in the bag from the closest center), we can significantly speed up the algorithm if we replace all entries by the closest integer power of $(1 + \delta)$, for some appropriately chosen δ. This will reduce the table size from (roughly) r^{tw} to $(\log_{(1+\delta)} r)^{\text{tw}}$.

In this way, if we can guarantee that these rounded values are always close to their exact counterparts that the original dynamic programming algorithm would use instead, we can obtain approximate solutions considerably faster. To that end, an abstract model of computation called the *Approximate Addition Tree* is introduced in [73] that captures these rounding ideas, while an analysis of their approximation performance guarantees that the rounding errors do not accumulate to an excessive extent.

Intuitively, an addition tree simulates the computations over a tree decomposition and the approximate version refers to the corresponding computations performed by the same tree, but on approximate values instead. The dynamic programming procedure for (k, r)-Center in fact computes the size k of the solution that we still want to exactly calculate, with approximate values used here for the distances of the vertices from the center-set. These values will be used as *state-representations* in the calculations of the algorithm, while the value of each entry of the DP table will contain the exact size of the corresponding (partial) solution. We will use these notions of Approximate Addition Trees to refer to the computations of distance/state-representation over the tree decomposition.[4]

Definition 36. *An* Addition Tree *is a full rooted binary tree T, where a non-negative integer* input *x_l is associated with each leaf l (provided with T) and a non-negative integer* value *y_v is associated with each node v. The value of each node is calculated as follows:*

1. *For each leaf l, we set $y_l \coloneqq x_l$;*
2. *For each internal node v with two children u_1, u_2, whose values have already been calculated, we set $y_v \coloneqq y_{u_1} + y_{u_2}$.*

An (exact) Addition Tree receives its inputs at the leaves, while the value of each node is calculated by simple addition of previously calculated values for its children. An Approximate Addition Tree operates in a similar way, with simple addition substituted

[4]The exposition of this section in the standalone version of [67] avoids making use of these definitions, but we give here a generic description as a similar approach is used in Section 4.3 for d-Scattered Set.

for a randomized variant, that in fact can only produce results rounded to a specific set of values.

Definition 37. *An* Approximate Addition Tree *with parameter δ is a full rooted binary tree T, where a non-negative integer* input x_l *is associated with each leaf l and a non-negative integer* approximate value z_v *is associated with each node v. The approximate value of each node is calculated as follows:*

1. *For each leaf l, we set $z_l \coloneqq x_l$;*
2. *For each internal node v with two children u_1, u_2, we set $z_v \coloneqq z_{u_1} \oplus z_{u_2}$, where the $\oplus$ operation is defined below.*

   Let $a_v \coloneqq z_{u_1} + z_{u_2}$. We call a_v the initial approximate value *of v. We use $\oplus$ to denote the following operation: for two non-negative numbers x_1, x_2 we define $x_1 \oplus x_2 \coloneqq 0$ if $x_1 = x_2 = 0$. Otherwise, we select a real number $\rho \in (0, 1)$ uniformly at random and set $x_1 \oplus x_2 \coloneqq (1+\delta)^{\lfloor \log_{(1+\delta)}(x_1+x_2)+\rho \rfloor}$.*

Observe that whenever an approximate value z_v is non-zero, it must be an integer power of $(1+\delta)$ and if the maximum value calculated by an exact tree would be at most polynomial in n, with $\delta = \Omega(1/\log^c n)$, there would be at most $\text{poly}(\log n)$ many different values that could appear in the approximate tree. This will allow for significantly smaller dynamic programming table sizes and a much faster algorithm.

Concerning the rounding error for each non-zero value for a node v of an Addition Tree (as for any node v with $y_v = 0$, we also have $z_v = 0$), with y_v its (positive) value and z_v its approximate value calculated if we view the tree as an Approximate Addition Tree, the *error* λ_v is defined as $\lambda_v \coloneqq \log_{(1+\delta)} \frac{z_v}{y_v}$. Note that λ_v and z_v are random variables (based on the random rounding choices), while y_v is fully specified once the inputs of the tree are fixed. We note that the randomized component of the technique is in fact not necessary here, as we could simply set $\rho = 0$ in the above definition and still get the same results[5]. The main theorem of [73] that we require to bound the maximum error in any appropriately constructed Approximate Addition Tree is given next.

Theorem 38 ([73], Theorem 3)**.** *If an Approximate Addition Tree has maximum height h, then for all nodes v we always have $|\lambda_v| \leq h + 1$. Therefore, if $\delta < \frac{\epsilon}{2h}$, then for all v we have $\max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} < 1 + \epsilon$.*

This is shown by induction on the maximum height h, the crucial observation being that the maximum absolute relative error never increases by more than a factor of $1 + \delta$.

For our purposes, the integers we would like to approximately store are the state-values $|s_j| \in [1, r]$, representing the maximum distance of vertex v_j in some bag X_i of the tree decomposition to some vertex selected in the center-set K, during the computation of the dynamic programming algorithm for the (k, r)-Center problem (see the end of this chapter for details). Instead of allowing use of all r exact values to signify the states, we will instead adopt probabilistically rounded approximations, that are inductively computed and propagated, while keeping accumulation of error bounded as well.

Let $\Sigma_\delta \coloneqq \{0\} \cup \{(1+\delta)^l | l \in \mathbb{N}\}$. Intuitively, Σ_δ is the set of rounded values that our modified algorithm may use to designate state-values. Of course, Σ_δ as defined is infinite, but we will only consider the set of values that are at most $(1+\epsilon) \cdot r$, denoted by Σ_δ^r. In

[5] Thus always rounding our values *down*, as in the application of the following chapter for Theorem 63.

this way, the size of Σ_δ^r is reduced to $\log_{1+\delta}((1+\epsilon)\cdot r)$, for an appropriate choice of δ, to be specified later. For all nodes i of the tree decomposition, we now have for a table entry $D_i[\sigma_0,\ldots,\sigma_t] : \left\{(\emptyset,0)\cup\{\uparrow\times\Sigma_\delta^{r-1}\}\cup\{\downarrow\times\Sigma_\delta^r\setminus\{0\}\}\right\}^{t+1}\mapsto\mathbb{N}^0\cup\{\infty\}$, with $0\leq t<\text{tw}$.

We next explain the inductive computation of the rounded states σ with $\pi\in\{\uparrow,\downarrow\}$, since for $\pi=\emptyset$ (zero state), it will be $|s|=|\sigma|=0$. First, for a vertex v_j with $\pi_j=\downarrow$ (negative state), consider a bag X_i introducing it and in particular the first computational case, where there must be a $v_{t'}\in X_i$ with $\pi_{t'}\in\{\downarrow,\emptyset\}$, such that $|s_{t'}|+d_i(v_{t'},v_j)\leq|s_j|$. The rounded state-value for v_j is then inductively given by $|\sigma_j|\coloneqq|\sigma_{t'}|\oplus d(v_{t'},v_{t+1})$, where $\oplus$ is the operator referred to in Definition 37. The base case of the inductive computation is vertex $v_{t'}$ being considered for inclusion in the center-set, that is $s_{t'}=(\emptyset,0)$.

On the other hand, for a vertex v_j with $\pi_j=\uparrow$ (positive state), we have $|\sigma_j|\coloneqq 0$ as a base case for the inductive computation of rounded state-values, while $\pi_j=\uparrow$, i.e. the "first" positive state that signifies this vertex will be at distance at most r to some vertex not yet introduced in the current stage of the algorithm. For the inductive step, we consider bag X_i forgetting vertex v_{t+1} and in particular the condition for inclusion of entries in modified table D^*, where there must be a $v_{t'}\in X_i$ with $\pi_{t'}=\uparrow$ and $|s_{t+1}|+d(v_{t'},v_{t+1})\leq|s_{t'}|$. Now, to appropriately define the inductive computation in a bottom-up manner, we identify vertex v_j with this $v_{t'}$ that appears in the bag forgetting vertex v_{t+1}, whose state-value we will increment to obtain the rounded state-value for v_j, i.e. the inductive computation is given by $|\sigma_j|\coloneqq|\sigma_{t+1}|\oplus d(v_j,v_{t+1})$. Note that this is the reason for the "inverted" scheme used for representation of positive state-values, as we would like the relative error to decrease with increasing values and propagation occurring alongside the progression of the algorithm over the tree decomposition and through the input graph.

Since our modified algorithm will make use of the rounded values $|\sigma_j|$, these being inductively computed for each vertex v_j by adding a previously calculated value $|\sigma_{t'}|$ for some neighbor $v_{t'}$ to their distance $d(v_j,v_{t'})$, a bound on the propagation of error would be required so we can be sure that any values we may use will not be too far from their exact counterparts. This is where Approximate Addition Trees become useful.

Before we go on, we require that the tree decompositions on which our algorithm is to be applied are rooted and of maximum height $O(\log n)$, for reasons that become apparent in the proof of Lemma 40. We use a result of [13] and Theorem 38 to bound the error of any value calculated in this way, based on an appropriate choice of δ and therefore set Σ_δ^r of available values.

Theorem 39. *[13] There is an algorithm which, given a tree decomposition of width tw of a graph on n nodes, produces a decomposition of the same graph with width at most* $3tw+2$ *and height* $O(\log n)$ *in polynomial time.*

Lemma 40. *Given* ϵ *and a tree decomposition* $(\mathcal{X},T)$ *with* $T=(I,F),\mathcal{X}=\{X_i|i\in I\}$, *where* T *is rooted, binary and of height* $O(\log n)$, *there exists a constant* C, *such that all rounded state-values* $|\sigma_j|\in\Sigma_\delta^r$ *will be within a* $(1+\epsilon)$ *range of their exact counterparts* $|s_j|$, *i.e.* $\frac{|s_j|}{(1+\epsilon)}<|\sigma_j|<(1+\epsilon)|s_j|,\forall v_j\in X_i,\forall j\in[0,|X_i|-1],\forall i\in I$, *where* $\delta=\frac{\epsilon}{C\log n}$.

Proof. Directly from its definition $|\sigma_j|\coloneqq|\sigma_{t'}|\oplus d(v_{t'},v_j)$, it is apparent that any calculated state-value for some vertex v_j is the same (or rather, follows the same distribution) as the value at the root of an Approximate Addition Tree T_j, whose children would be an Approximate Addition Tree $T_{t'}$ computing the value for vertex $v_{t'}$ appearing in the same bag and a leaf with value equal to the distance $d(v_j,v_{t'})$. This is true for all state-values

we will require, the base cases being states $\sigma_j = (\emptyset, 0)$ (negative) and simply $\sigma_j = (\uparrow, 0)$ (positive).

Next, observe that during the computations of the algorithm, the maximum height h of any such Approximate Addition Tree T_j can only increase by one if some vertex is introduced in the tree decomposition, as paths to and from it become available. This means no such Approximate Addition Tree T_j can be of height larger than the height of the tree decomposition T and thus we have $h = c \log n$ for some constant c. By Theorem 38, there exists some constant $C > 2c$, such that if $\delta = \frac{\epsilon}{C \log n}$, we always have for all nodes v of any such Approximate Addition Tree that $\max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} < 1 + \epsilon$. Now, $z_v = |\sigma_j|$ is the rounded state-value calculated for some vertex v_j by this node v of the Approximate Addition Tree T_j and $y_v = |s_j|$ is the exact value calculated by the same node for the same vertex if T_j was seen as an exact Addition Tree. This indeed gives $\frac{|s_j|}{(1+\epsilon)} < |\sigma_j| < (1+\epsilon)|s_j|, \forall v_j \in X_i, \forall j \in [0, |X_i| - 1]$. □

Correctness: Naturally, our modified algorithm making use of these rounded values to represent states will not perform the same computations as the exact version. The new statement of correctness, taking into account the approximate values now computed, is the following (comparison with the simpler statement of the basic dynamic programming given at the end of the chapter may be of interest):

$$\forall i \in I, \forall (\sigma_0, \ldots, \sigma_t) \in \Big\{(\emptyset, 0) \cup \{\uparrow \times \Sigma_\delta^{r-1}\} \cup \{\downarrow \times \Sigma_\delta^r \setminus \{0\}\}\Big\}^{t+1}, 0 \leq t < \text{tw} : \tag{3.15}$$

$$\Big\{\{D_i[\sigma_0, \ldots, \sigma_t] = \min_{K \subseteq V_i \setminus X_i} |K| : \tag{3.16}$$

$$\{\forall u \in V_i \setminus X_i : \Big\{\exists v \in K \wedge \exists T : z_{\text{root}(T)} \sim |\sigma| \wedge y_{\text{root}(T)} \leq r \wedge d_i(u, v) \leq |\sigma|\Big\} \vee \tag{3.17}$$

$$\vee \Big\{\exists v_j \in X_i : \sigma_j = (\emptyset, 0) \wedge \exists T : z_{\text{root}(T)} \sim |\sigma| \wedge y_{\text{root}(T)} \leq r \wedge d_i(u, v_j) \leq |\sigma|\Big\} \vee \tag{3.18}$$

$$\vee \Big\{\exists v_j \in X_i : \pi_j = \uparrow \wedge \exists T : z_{\text{root}(T)} \sim |\sigma_j| \wedge d_i(u, v_j) \leq y_{\text{root}(T)}\Big\}\} \bigwedge \tag{3.19}$$

$$\bigwedge \{\forall j \in [0, t] : \pi_j = \downarrow \Rightarrow \Big\{\exists u \in K, \exists T' : z_{\text{root}(T')} \sim |\sigma_j| \wedge d_i(u, v_j) \leq y_{\text{root}(T')}\Big\} \vee \tag{3.20}$$

$$\vee \Big\{\exists v_{t'} \in X_i, \exists T'' : \pi_{t'} = \emptyset \wedge z_{\text{root}(T')} \sim |\sigma_j| \wedge (d_i(v_j, v_{t'}) \leq y_{\text{root}(T')})\}\Big\}\} \tag{3.21}$$

$$\vee D_i[\sigma_0, \ldots, \sigma_t] = \infty\Big\}. \tag{3.22}$$

In words, the above states that for every node i and all possible rounded-state-configurations $(\sigma_0, \ldots, \sigma_t)$ (3.15), table entry $D_i[\sigma_0, \ldots, \sigma_t]$ contains the minimum size of a subset K of $V_i \setminus X_i$ (3.16), such that for every vertex u in the subgraph G_i but not in X_i, there is a vertex v included in K and an Approximate Addition Tree T, such that the root of T computes the same value $z_{\text{root}(T)}$ as some[6] state-value $|\sigma|$ (following the same distribution, denoted by $\sim$) and if T was seen as an exact Addition tree, the distance from u to v is at most $|\sigma|$ (3.17), or there is a vertex v_j in the current bag with $\sigma_j = (\emptyset, 0)$ (zero state) and an Approximate Addition Tree T, such that the root of T computes the same value $z_{\text{root}(T)}$ as some state-value $|\sigma|$ and if T was seen as an exact Addition tree, the distance from u to v_j is at most $|\sigma|$ (3.18), or there is a vertex v_j in the current bag with $\pi_j = \uparrow$ (positive state) and an Approximate Addition Tree T, such that the root of

[6]This is the state-value for some vertex already forgotten in the current stage of the algorithm.

T computes the same value $z_{\text{root}(T)}$ as $|\sigma_j|$ and if T was seen as an exact Addition Tree, the distance from u to v_j is at most the value $y_{\text{root}(T)}$ computed at the root of T (3.19), *and* for every vertex v_j in X_i, negative state with $\pi_j = \downarrow$ implies there is a vertex u in K and an Approximate Addition Tree T', such that the root of T' computes the same value $z_{\text{root}(T')}$ as $|\sigma_j|$ and if T' was seen as an exact Addition Tree, the distance from u to v_j is at most the value $y_{\text{root}(T')}$ computed at the root of T' (3.20), or $\pi_j = \downarrow$ implies that there is a vertex $v_{t'}$ in X_i with $|\pi_{t'}| = 0$ and an Approximate Addition Tree T', such that the root of T' computes the same value $z_{\text{root}(T')}$ as $|\sigma_j|$ and if T' was seen as an exact Addition Tree, the distance from v_j to $v_{t'}$ is at most the value $y_{\text{root}(T')}$ computed at the root of T' (3.21), or if there is no such $K \subseteq V_i$, we have $D_i[\sigma_0, \ldots, \sigma_t] = \infty$ for this table entry (3.22), representing invalidity of the corresponding partial solution.

The main difference between the above and the exact statement of correctness is the requirement for the existence of (Approximate) Addition Trees, whose values at the root follow the same distribution as the inductively computed state-values, while if these were seen as exact Addition Trees, their values at the root would need to satisfy the distance requirements, as in the exact computation. All arguments for correctness of the algorithm pertaining to its inner mechanism directly transfer here for the modified statement given above, keeping in mind that now state-value computation for some vertex v_j is given by $|\sigma_j| \coloneqq |\sigma_{t'}| \oplus d(v_{t'}, v_j)$, where $|\sigma_{t'}|$ is a previously calculated value for some vertex $v_{t'}$ appearing in the same bag.

Lemma 41. *All vertices are at distance at most* $(1+\epsilon)\cdot r$ *from some vertex in the center-set* K *output by the algorithm, if the above statement of correctness (3.15-3.22) holds.*

Proof. For a given node i, if the statement of correctness (3.15-3.22) holds, then (3.17) states that for all vertices u strictly before the current bag, there must be some vertex v already included in K and some Approximate Addition Tree T computing a value $z_{\text{root}(T)}$ that follows the same distribution as some state-value $|\sigma|$ the algorithm had earlier computed for this vertex v, while also letting $|s|$ denote the state-value the exact dynamic program would have assigned to vertex v instead. Now, if T was an exact Addition Tree, the value $y_{\text{root}(T)}$ it computes would be required to be $\leq r$. As the distance between u and v is required to be $d_i(u, v) \leq |\sigma|$ and by Lemma 40 we have $|\sigma| < (1 + \epsilon) \cdot |s|$, we also get $d_i(u, v) < (1 + \epsilon) \cdot |s| \leq (1 + \epsilon) \cdot r$, which is precisely the approximate version (i.e. with a multiplicative $1 + \epsilon$ error factor) of the corresponding line in the statement of correctness of the exact dynamic program. Similar arguments can be made for all lines of the statement of correctness (3.17-3.21), showing that all distances are indeed within a $(1 + \epsilon)$ factor of their exact counterparts. □

Theorem 42. *There is an algorithm which, given a weighted instance of* (k, r)-CENTER $[G, k, r]$*, a tree decomposition of* G *of width* tw *and a parameter* $\epsilon > 0$*, runs in time* $O^*((tw/\epsilon)^{O(tw)})$ *and either returns a* $(k, (1 + \epsilon)r)$*-center of* G*, or correctly concludes that* G *has no* (k, r)*-center.*

Proof. First, according to the statement of Lemma 40, we select $\delta = \frac{\epsilon}{C \log n}$, that we use to define the set Σ^r_δ. Then, based on the modified state representations σ_j, with their inductive computation as described above, we use the dynamic programming algorithm of running-time $(2r + 1)^{\text{tw}} \cdot n^{O(1)}$ for the (k, r)-CENTER problem on the bounded-height transformation of the given nice tree decomposition of width tw.

Correctness of the algorithm and justification of the approximation bound are given above (Lemma 41), while the running-time crucially depends on the size of the set of rounded values $|\Sigma_\delta^r| = \log_{1+\delta}((1+\epsilon)\cdot r) = \frac{\log((1+\epsilon)\cdot r)}{\log(1+\frac{\epsilon}{C\log n})} = \frac{\log((1+\epsilon)\cdot r)}{\frac{\epsilon}{C\log n}} \leq \frac{O(\log n)}{\epsilon}$, where we used the approximation $\log(1+x) \approx x$ for $x \approx 0$, as well as Lemma 2. □

3.4 Clique-width revisited: FPT approximation scheme

We give here an FPT-AS for (k,r)-Center parameterized by cw, both for unweighted and for weighted instances (for a weighted definition of cw which we explain below). Our algorithm builds on the algorithm of Subsection 3.3, and despite the added generality of the parameterization by cw, we are able to obtain an algorithm with similar performance: for any $\epsilon > 0$, our algorithm runs in time $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$ and produces a $(k,(1+\epsilon)r)$-center if the input instance admits a (k,r)-center.

Our main strategy, which may be of independent interest, is to pre-process the input graph $G=(V,E)$ in such a way that the answer does not change, yet producing a graph whose tw is bounded by $O(\mathrm{cw}(G))$. The main insight that we rely on, which was first observed by [54], is that a graph of low cw can be transformed into a graph of low tw if one removes all large bi-cliques. Unlike previous applications of this idea (e.g. [74]), we do not use the main theorem of [54] as a "black box", but rather give an explicit construction of a tree decomposition, exploiting the fact that (k,r)-Center allows us to relatively easily eliminate complete bi-cliques. As a result, we obtain a tree decomposition of width not just bounded by some function of $\mathrm{cw}(G)$, but actually $O(\mathrm{cw}(G))$.

In the remainder we deal with the weighted version of (k,r)-Center. To allow clique-width expressions to handle weighted edges, we interpret the clique-width join operation η as taking three arguments. The interpretation is that $\eta(a,b,w)$ is an operation that adds (directed) edges from all vertices with label a to all vertices with label b and gives weight w to all these edges. It is not hard to see that if a graph has a (standard) clique-width expression with cw labels, it can also be constructed with cw labels in our context, if we replace every standard join operation $\eta(a,b)$ with $\eta(a,b,1)$ followed by $\eta(b,a,1)$. Hence, the algorithm we give also applies to unweighted instances parameterized by (standard) clique-width.

We will also deal with a generalization of (k,r)-Center, where we are also supplied, along with the input graph $G=(V,E)$, a subset $I \subseteq V$ of *irrelevant* vertices. In this version, a (k,r)-center is a set $K \subseteq V \setminus I$, with $|K|=k$, such that all vertices of $V \setminus I$ are at distance at most r from K. Clearly, the standard version of (k,r)-Center corresponds to $I=\emptyset$. As we explain in the proof of Theorem 45, this generalization does not make the problem significantly harder.

In addition to the above, in this subsection we will allow edge weights to be equal to 0. This does not significantly alter the problem, however, if we are interested in approximation and allow r to be unbounded, as the following lemma shows:

Lemma 43. *There exists a polynomial algorithm which, for any $\epsilon > 0$, given an instance $I=[G,w,k,r]$ of (k,r)-Center, with weight function $w: V \to \mathbb{N}$, produces an instance $I'=[G,w',k,r']$ on the same graph with weight function $w': V \to \mathbb{N}^+$, such that we have the following: for any $\rho \geq 1$, any $(k,\rho r')$-center of I' is a $(k,\rho r)$-center of I; any $(k,\rho r)$-center of I is a $(k,(1+\epsilon)\rho r')$-center of I'.*

Proof. We define a scaling factor $B := \frac{n}{\epsilon}$. We set $w'((u,v)) = B \cdot w((u,v)) + 1$, for all $(u,v) \in E$. We set $r' = B \cdot r$. Suppose that we have a $(k, \rho r)$-center K for w. We use the same solution for w' and the maximum distance from K to any vertex is at most $B\rho r + n = \rho r' + n$. However, $n \leq \epsilon r'$, hence this solution has maximum distance at most $(1+\epsilon)\rho r'$. For the converse direction, suppose that we have a solution for the function w' with maximum distance $\rho r'$. We use the same solution and now the cost is at most $\rho r'/B \leq \rho r$, since $w((u,v)) \leq w'((u,v))/B$. □

Our main tool is the following lemma, whose strategy is to replace every large label-set by two "representative" vertices, in a way that retains the same distances among all vertices of the graph. Applying this transformation repeatedly results in a graph with small treewidth. The main theorem of this subsection then follows from the above.

Lemma 44. *Given a (k,r)-*Center *instance $G = (V,E)$ along with a clique-width expression T for G on cw labels, we can in polynomial time obtain a (k,r)-*Center *instance $G' = (V', E')$ with $V \subseteq V'$, and a tree decomposition of G' of width $tw = O(cw)$, with the following property: for all k, r, G has a (k,r)-center if and only if G' has a (k,r)-center.*

Proof. Suppose we are given a clique-width expression of G, represented as a binary tree T, using cw labels. We will first construct a new clique-width expression T', using $\mathrm{cw}+3$ labels in a way that preserves all (k,r)-centers. The end result will be that T' does not contain any join operations involving large label-sets. We will use this property to construct a tree decomposition of the resulting graph.

Let c be an appropriate small constant (for concreteness, let $c = 10$). We will say that a label $l \in \{1, \ldots, \mathrm{cw}\}$ is *big* in a node $x \in T$, if the graph described by the sub-expression rooted at x has at least c vertices with label l. We say that l is *newly big* in $x \in T$, if l is big in x but it is not big in any of the children of x in T. Finally, we say that l is *active inside* (respectively *active outside*) in $x \in T$ if l is newly big in x and, furthermore, there exists an arc $e \in E$ that is not yet present in x, whose head (respectively tail) is incident on a vertex with label l in x. In other words, a label is active (inside or outside) in a sub-expression, if it contains many vertices and there is a join operation (of the respective direction) that is applied to its vertices somewhere higher in T.

Suppose that the nodes of T are ordered in some arbitrary way, so that a child is ordered before its parent. Our transformation is the following: as long as there exists $l \in \{1, \ldots, \mathrm{cw}\}$ and $x \in T$ such that l is active (inside or outside) in x, we select the first such x in the ordering and let l be an active label in x. We insert directly above x in T the following operations: if l is active inside we introduce a vertex s with label $\mathrm{cw}+1$, and if l is active outside we introduce a vertex t with label $\mathrm{cw}+2$; we then add the operations $\eta(\mathrm{cw}+1, l, 0)$, $\eta(l, \mathrm{cw}+2, 0)$, followed by the rename operations $\rho(l, \mathrm{cw}+3), \rho(\mathrm{cw}+1, l), \rho(\mathrm{cw}+2, l)$. In words, this transformation (potentially) adds a "source" s, or a "sink" t to the graph (or both), gives a junk label $\mathrm{cw}+3$ to all vertices that previously had label l in x, and uses s, t as the new representatives of this class of vertices. See also Figure 3.11 for an illustration. We repeat this process until no $x \in T$ contains an active label; this finishes in polynomial time because once we eliminate all active labels from a node x, we do not re-visit it. If the original instance had a set of irrelevant vertices $I \subseteq V$, we construct a set of irrelevant vertices I' for G' by adding to I all the s, t vertices created in the above procedure.

Let us first argue that any (k,r)-center of the original instance can be transformed into a (k,r)-center of the new instance. Consider one step of the above procedure applied on a node $x \in T$. The main observation is that such a step does not change the distance

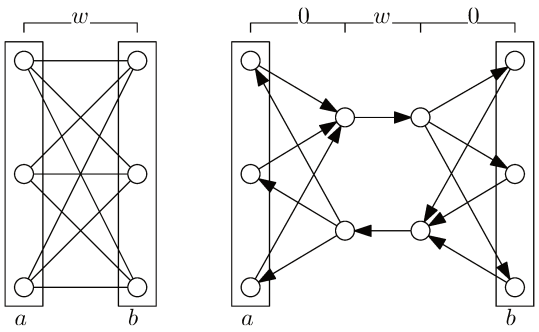


Figure 3.11: A simplified demonstration of our bi-clique transformation.

between any two vertices $u, v \in V$ in the final graph. Suppose u has label l in x. Any edge incident on u in G that was removed as a result of this operation does not yet appear in x and hence can be replaced by a path of the same cost through s or t. Similarly, any u, v path in the new graph that uses s or t in the new graph, must also use an edge (that is not present in x) which was removed by the operation. Such a path can be replaced by this edge in G.

As a result of the above argument, all distances between vertices of V are preserved after the end of the procedure. Since all vertices of $V' \setminus V$ are irrelevant in G', any (k, r)-center of G is also a (k, r)-center of G' and vice-versa, as any solution can ignore these vertices and no solution can contain any of them.

To see that G' has small treewidth, observe that we now have a clique-width expression for G' that uses $\mathrm{cw} + 3$ labels and has the following property: for any join operation $\eta(a, b, w)$ applied on a node x, there are at most $2c$ vertices with labels a or b in x. To see this, observe that this operation is either an operation that was originally in T, or an operation added during the transformation procedure. In the former case, neither label is big in x, otherwise, there would be a descendant of x where a label is active, and we would have applied the operation. In the latter case, there is a single vertex with label b, and a was newly active, hence it cannot contain more than $2c - 2$ vertices (if it was the result of a rename or union operation on two previously small label-sets).

We can therefore construct a tree decomposition of G' by taking its clique-width expression T' and making a bag for each of its nodes. For the bag B_x that corresponds to node x of T', we find all labels $l \in \{1, \ldots, \mathrm{cw} + 3\}$ which are small in x (that is, there are at most $2c$ vertices of this label) and place all vertices with that label in x in B_x. This is a valid tree decomposition because: (i) as argued above all join operations involve two small label-sets, hence any edge of the graph is contained in some bag, (ii) if $u \in B_x$ but $u \notin B_y$, for B_y being the parent of B_x, we know that u also does not appear in any ancestor of B_y, because once a vertex is part of a large label, it remains so and it does not appear in the other child of B_y (if it exists), or its descendants, because if B_y has two children it must correspond to a disjoint union operation in T'. To complete the proof we observe that we have placed at most $2c(\mathrm{cw} + 3)$ vertices in each bag. □

Theorem 45. *Given $G = (V, E)$, $k, r \in \mathbb{N}^+$, clique-width expression T for G on cw labels and $\epsilon > 0$, there exists an algorithm that runs in time $O^*((cw/\epsilon)^{O(cw)})$ and either produces a $(k, (1 + \epsilon)r)$-center, or correctly concludes that no (k, r)-center exists.*

Proof. Given the (k, r)-Center instance, we use Lemma 44 to obtain an instance G'

and its tree decomposition of width $O(\text{cw})$. We then invoke Lemma 43 to make all edge weights strictly positive, and then use the algorithm of Subsection 3.3. We remark that this algorithm can be easily adjusted to handle the more general case of the problem where some vertices are irrelevant: we simply need to modify the computation on Forget nodes to allow the algorithm to retain a solution that has not satisfied a vertex u if u is irrelevant and the computation on Introduce nodes to disregard a solution that considers u for inclusion in the center-set. □

Basic DP algorithm for Treewidth

We describe the basic dynamic programming algorithm for the decision version of the (k, r)-Center problem (weighted/unweighted) on a given tree decomposition. This result is not novel and is given here only for completeness. The algorithm's complexity is $O^*(2^{\text{tw}}(2r+1)^{\text{tw}})$ and the multiplicative factor of 2^{tw} has not been avoided (as in [22]), since an application of the subset convolution technique is not necessary for the exposition of the approximation algorithm given in Section 3.3.

Recall that the input is a graph $G = (V, E)$, along with two numbers $k, r \in \mathbb{N}^+$, denoting the size of the requested center-set $K \subseteq V$, and the maximum distance from any vertex to some vertex included in K, respectively. A nice tree decomposition $(\mathcal{X}, T)$ of minimum width tw for G is also assumed to be given as input, where $T = (I, F)$ is a tree and $\mathcal{X} = \{X_i | i \in I\}$ is the set of bags, while $\max_{i \in I} |X_i| - 1 = \text{tw}$.

Table description: There is a table D_i associated with every node $i \in I$ of the tree decomposition with $X_i = \{v_0, \ldots, v_t\}$, $0 \leq t \leq \text{tw}$, while each table entry $D_i[s_0, \ldots, s_t]$ is indexed by a $t+1$-sized tuple $(s_0, \ldots, s_t)$ of *state-configurations*, assigning a state $s_j \in \{(\emptyset, 0), (\downarrow, 1), \ldots, (\downarrow, r), (\uparrow, 0), \ldots, (\uparrow, r-1)\}$ to each vertex v_j, $\forall j \in [0, t]$. Note that these are couples of a *direction* (denoted $\pi_j \in \{\emptyset, \uparrow, \downarrow\}$) and a *value* $|s_j|$ from 0 to r (or $r-1$). There are $2r+1$ possible states for each vertex, designating its distance to a center-set K at each stage of the algorithm:

- *Negative* state $s_j \in \{(\downarrow, 1), \ldots, (\downarrow, r)\}$ signifies vertex v_j is at distance at most $|s_j|$ from a selected vertex $u \in K$ within the terminal subgraph G_i defined by node i, i.e. $\exists u \in K \cap V_i : d_i(u, v_j) \leq |s_j|$.

- *Zero* state $s_j = (\emptyset, 0)$ signifies vertex v_j is considered for selection in the center-set, i.e. $v_j \in K$.

- *Positive* state $s_j \in \{(\uparrow, 0), (\uparrow, 1), \ldots, (\uparrow, r-1)\}$ signifies vertex v_j is at distance at most $r - |s_j|$ from a vertex u that has not yet been introduced and will be selected in K at a later stage of the algorithm (being "expected"), i.e. $\exists u \in K \cap \{V \setminus V_i\} : d(u, v_j) + |s_j| \leq r$.

Note that for positive states s_j, the interpretation of their values $|s_j|$ is, after a fashion, the "inverse" of what negative values signify (i.e. maximum distance from a center-vertex): the informal meaning here is that there have already been $|s_j|$ "steps taken" for connection of vertex v_j to a vertex in the center-set K. The reasons for this inversion of significance of positive state-values become apparent in Section 3.3, where approximate values are inductively computed and relative error should decrease with increasing values.

For a node $i \in I$, each table entry $D_i[s_0, \ldots, s_t]$ contains the minimum number of vertices a center-set $K \subseteq V_i \setminus X_i$ will require to cover all vertices in the terminal subgraph G_i, excluding those in the current bag X_i, their own situation being described by the particular state configuration $(s_0, \ldots, s_t)$ indexing this entry. The computation of each entry is based on the type of node the table is associated with (leaf, introduce, forget, or join), previously computed entries of the table associated with the preceding node(s) and the structure of the node's terminal subgraph. In particular, we have $\forall i \in I, D_i[s_0, \ldots, s_t] : \{(\emptyset, 0), (\downarrow, 1), \ldots, (\downarrow, r), (\uparrow, 0), \ldots, (\uparrow, r-1)\}^{t+1} \mapsto \mathbb{N}^0 \cup \{\infty\}$, where $0 \leq t < \text{tw}$ and ∞ is an arbitrarily large integer, signifying that state-configuration $(s_0, \ldots, s_t)$ does not correspond to a valid partial solution (in conjunction with other relevant information). The inductive computation of all table entries for each type of node is given next.

Leaf node i with $X_i = \{v_0\}$: We have:

$$D_i[s_0] \coloneqq \begin{cases} \infty, & \text{if } \pi_0 = \downarrow; \\ 0, & \text{otherwise.} \end{cases}$$

Leaf nodes contain only one vertex and all entries for negative states are assigned an ∞ value, as the partial solutions defined are invalid in the absence of potential connections, while zero and positive states are initialized to 0.

Introduce node i with $X_i = X_{i-1} \cup \{v_{t+1}\}$: Given state-configuration $(s_0, \ldots, s_t)$, assume (without loss of generality) that for some $t' \in [0, t]$ (if any) we have $\pi_j = \uparrow, \forall j \in [0, t']$, while $\pi_{t'+1}, \ldots, \pi_t \in \{\emptyset, \downarrow\}$. Let $\mathcal{S}^+_{t'}$ denote the set of all possible tuples $S = (s^+_0, \ldots, s^+_{t'})$, where each state s^+_j is either the same state s_j, or its opposite (in terms of $\uparrow, \downarrow$ direction):

$$\mathcal{S}^+_{t'} \coloneqq \{(s^+_0, \ldots, s^+_{t'}) | \forall j \in [0, t'] : \{(s^+_j = s_j) \vee (\pi^+_j = \downarrow \wedge |s^+_j| = r - |s_j|)\}\}.$$

Then we have:

$$D_i[s_0, \ldots, s_t, s_{t+1}] \coloneqq \begin{cases} D_{i-1}[s_0, \ldots, s_t], & \text{if } \pi_{t+1} = \downarrow \text{ and} \\ & \exists v_{t'} \in X_i \text{ with } \pi_{t'} \in \{\downarrow, \emptyset\}, \\ & \text{such that } |s_{t'}| + d_i(v_{t'}, v_{t+1}) \leq |s_{t+1}|; \\ \infty, & \text{if } \pi_{t+1} = \downarrow \text{ and} \\ & \forall v_{t'} \in X_i \text{ with } \pi_{t'} \in \{\downarrow, \emptyset\}, \\ & \text{it is } |s_{t'}| + d_i(v_{t'}, v_{t+1}) > |s_{t+1}|; \\ \min_{S \in \mathcal{S}^+_{t'}} \{D_{i-1}[S, s_{t'+1}, \ldots, s_t]\}, & \text{if } \pi_{t+1} = \emptyset \text{ and} \\ & \forall j \in [0, t'] \text{ with } \pi_j = \uparrow, \\ & \text{it is } r - |s_j| \leq d_i(v_j, v_{t+1}); \\ D_{i-1}[s_0, \ldots, s_t], & \text{if } \pi_{t+1} = \uparrow . \end{cases}$$

When a vertex v_{t+1} is introduced, all entries of the previous table need to be updated in accordance with every possible state s_{t+1}, while respecting state concurrences imposed by (potential) connectivity between v_{t+1} and other vertices present in X_i. For state-configurations of the new table where $\pi_{t+1} = \downarrow$, there must be some vertex $v_{t'}$ in X_i with

$\pi_{t'} = \downarrow$, such that their states "fit" their distance, or $|s_{t'}| + d(v_{t'}, v_{t+1}) \leq |s_{t+1}|$. For state-configurations with $\pi_{t+1} = \emptyset$, entries of the previous table where vertices v_j with $r - |s_j| \leq d_i(v_j, v_{t+1})$ had a positive state might now correspond to partial solutions, in the entries for which, these vertices' states are "flipped", since v_{t+1} is now included in the bag (and not just "expected"), thus the minimum value of all suitable entries is selected. Note that $\mathcal{S}^+_{t'}$ includes only these vertices v_j with $\pi_j = \uparrow$, such that $r - |s_j| \leq d_i(v_j, v_{t+1})$, while other vertices v_l with $\pi_l = \uparrow$ and non-fitting values might be included in $v_{t'+1}, \ldots, v_t$. For entries with $\pi_{t+1} = \uparrow$, all partial solutions naturally extend those computed for the previous node and their values can be directly transferred.

Forget node i with $X_i = X_{i-1} \setminus \{v_{t+1}\}$: Let $D^*_{i-1}[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \uparrow\}]$ denote entries for which $\exists v_{t'} \in X_i$ with $\pi_{t'} = \uparrow$ and $|s_{t+1}| + d_i(v_{t'}, v_{t+1}) \leq |s_{t'}|$. Then we have:

$$D_i[s_0, \ldots, s_t] \coloneqq \min\{D_{i-1}[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \downarrow\}], \\ D_{i-1}[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \emptyset\}] + 1, D^*_{i-1}[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \uparrow\}]\}.$$

Our algorithm is designed so as not to count the number of vertices considered for selection within each bag (vertices with zero states) in the computed value for each entry and the crucial increase of $+1$ for appropriate entries is not performed during computation of introduce nodes' tables. Instead, our algorithm increases the count of selected vertices only when a vertex v_{t+1} is forgotten and only in case a partial solution with a corresponding state-configuration that assigns $s_{t+1} = (\emptyset, 0)$ is selected as the most preferable choice for extracting a table for the new bag, that is one dimension lower than the previous bag's table, due to removal of the forgotten vertex from consideration.

One may note this makes our algorithm seemingly prefer partial solutions where selection of vertices happens at the latest possible stage, since out of two partial solutions of otherwise equal cost, our algorithm will choose the one where the selected vertex remains in the bag, owing to the $+1$ within the operation of comparison for identifying the minimum, yet this approach can be justified as a more conservative way of administering the (limited) number of possible selections for inclusion in the center-set.

Join node i with $X_i = X_{i-1} = X_{i-2}$: In contrast to the computation of introduce nodes, given state-tuple $(s_0, \ldots, s_t)$, we assume (without loss of generality) that for some $t' \in [0, t]$ (if any) it is $\pi_j = \downarrow, \forall j \in [0, t']$, while $\pi_{t'+1}, \ldots, \pi_t \in \{\emptyset, \uparrow\}$. Now, let $\mathcal{S}^-_{t'}$ denote the set of all possible tuples $S = (s^-_0, \ldots, s^-_{t'})$, where each state s^-_j is either the same state s_j, or its opposite (in terms of $\uparrow, \downarrow$ direction):

$$\mathcal{S}^-_{t'} \coloneqq \{(s^-_0, \ldots, s^-_{t'}) | \forall j \in [0, t'] : \{(s^-_j = s_j) \vee (\pi^-_j = \uparrow \wedge |s^-_j| = r - |s_j|)\}\}.$$

and for some tuple $S \in \mathcal{S}^-_{t'}$ let $\bar{S}$ denote the complementary tuple (where the state of each vertex is likewise reversed). Then we have:

$$D_i[s_0, \ldots, s_t] \coloneqq \min_{S \in \mathcal{S}^-_{t'}} \{D_{i-1}[S, s_{t'+1}, \ldots, s_t] + D_{i-2}[\bar{S}, s_{t'+1}, \ldots, s_t]\}.$$

For join nodes the bags of both children contain the same set of vertices, yet the partial solutions characterized by the entries of each table concern distinct terminal subgraphs G_{i-1} and G_{i-2}. For state-configurations in which no vertex is assigned a negative state, a simple addition of the two entries from each table for this state-configuration is sufficient, as

the entries' values refer to partial solutions for disjoint parts of the graph and the number of selected vertices currently in the bag is not counted. For state-configurations with strictly negative states for some vertices there are two options for connection per negative state s_j, each signifying connection through one of the two subgraphs, with directions $\uparrow, \downarrow$ denoting inclusion-in/exclusion-from each of the two terminal subgraphs. The addition of complementary entries from the two tables (for all compatible pairs) is thus needed to assemble a comprehensive account for both subgraphs, with subsequent selection of the overall minimum value. It is precisely this aspect of the considerations usually involved in the computation of join nodes' tables that creates an additional factor in the complexity of dynamic programming procedures of this type. Optimally performing these computations (with running-time matching space requirements) is not too hard to accomplish via an extension of the fast subset convolution technique for appropriately partitioned subsets.

Correctness: To show correctness of the algorithm we need to establish that for every node $i \in I$, each table entry $D_i[s_0, \ldots, s_t]$ contains a partial solution for the sub-problem as restricted to G_i, i.e. the minimum number of vertices to select in K from $V_i \setminus X_i$, such that every vertex in $V_i \setminus X_i$ is at distance at most r from a vertex in K or a vertex $v_j \in X_i$ with zero state $s_j = (\emptyset, 0)$, or at distance at most $|s_j|$ from a vertex $v_j \in X_i$ with $\pi_j = \uparrow$, while for all vertices in X_i, their state for this entry describes their situation in this partial solution. In particular, we need to show the following:

$$\forall i \in I, \forall (s_0, \ldots, s_t) \in \{(\emptyset, 0), (\downarrow, 1), \ldots, (\downarrow, r), (\uparrow, 0), \ldots, (\uparrow, r-1)\}^{t+1}, 0 \leq t < \text{tw} : \tag{3.23}$$

$$\Big\{ \{D_i[s_0, \ldots, s_t] = \min_{K \subseteq V_i \setminus X_i} |K| : \tag{3.24}$$

$$\{\forall u \in V_i \setminus X_i : \{\exists v \in K : d_i(u, v) \leq r\} \vee \tag{3.25}$$

$$\vee \{\exists v_j \in X_i : \pi_j = \emptyset \wedge d_i(u, v_j) \leq r\} \vee \tag{3.26}$$

$$\vee \{\exists v_j \in X_i : \pi_j = \uparrow \wedge d_i(u, v_j) \leq |s_j|\}\} \bigwedge \tag{3.27}$$

$$\bigwedge \{\forall j \in [0, t] : \pi_j = \downarrow \Rightarrow \{\exists u \in K : d_i(u, v_j) \leq |s_j|\} \vee \tag{3.28}$$

$$\vee \{\exists v_{t'} \in X_i : (\pi_{t'} = \emptyset) \wedge (d_i(v_j, v_{t'}) \leq |s_j|)\}\}\} \tag{3.29}$$

$$\vee D_i[s_0, \ldots, s_t] = \infty \Big\}. \tag{3.30}$$

In words, the above states that for every node i and all possible state-configurations $(s_0, \ldots, s_t)$ (3.23), table entry $D_i[s_0, \ldots, s_t]$ contains the minimum size of a subset K of $V_i \setminus X_i$ (3.24), such that for every vertex u in the subgraph G_i but not in X_i, there is a vertex v included in K whose distance is at most r from u (3.25), or there is a vertex v_j in the current bag with $\pi_j = \emptyset$ (zero state) whose distance from u is at most r (3.26), or there is a vertex v_j in the current bag with $\pi_j = \uparrow$ (positive state) whose distance from u is at most $|s_j|$ (3.27), *and* for every vertex v_j in X_i, negative state with $\pi_j = \downarrow$ implies there is a vertex u in K whose distance from v_j is at most $|s_j|$ (3.28), or $\pi_j = \downarrow$ implies that there is a vertex $v_{t'}$ in X_i with state $|s_{t'}| = 0$ whose distance from v_j is at most $|s_j|$ (3.29), or if there is no such $K \subseteq V_i$, we have $D_i[s_0, \ldots, s_t] = \infty$ for this table entry (3.30). This is shown by induction on the nodes $i \in I$:

- Leaf node i with $X_i = \{v_0\}$: This is the base case of our induction. Observe that for $\pi_0 = \downarrow$, since $V_i \setminus X_i = \emptyset$, there is no $K \subseteq V_i \setminus X_i$ for which (3.25-3.29) is true and

$D_i[s_0] = \infty$ (3.30), while for $\pi_0 \in \{\emptyset, \uparrow\}$, value 0 is the correct minimum size for any such K (3.24). In the following cases, we assume (our induction hypothesis) that all entries of D_{i-1} (and D_{i-2} for join nodes) either contain the correct minimum value for some $K \subseteq V_{i-1} \setminus X_{i-1}$ (3.24), or have an ∞ value (3.30).

- Introduce node i with $X_i = X_{i-1} \cup \{v_{t+1}\}$: For entries $D_i[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \downarrow\}]$, since introducing a vertex with a negative state requires only inspection of partial solutions for potential integration of this vertex, the correct minimum sizes of $K \subseteq V_i \setminus X_i$ to cover all $u \in V_i \setminus X_i$ (or ∞) are computed in $D_{i-1}[s_0, \ldots, s_t]$ (3.25-3.27), while for vertices v_j, with $j \in [0, t]$ (already in X_i) and $\pi_j = \downarrow$ we know there is either a $u \in K$ with $d_i(u, v_j) \leq |s_j|$ (3.28), or a $v_{t'} \in X_{i-1}$ (and also X_i) with $s_{t'} = \{0, \emptyset\}$ and $d_i(v_j, v_{t'}) \leq |s_j|$ (3.29).

  To show the same for v_{t+1}, observe that for entries where there exists a vertex $v_{t'}$ in X_i with $\pi_{t'} \in \{\downarrow, \emptyset\}$ and $|s_{t'}| + d_i(v_{t'}, v_{t+1}) \leq |s_{t+1}|$, the correct value is computed in $D_{i-1}[s_0, \ldots, s_{t'}, \ldots, s_t]$, while for entries $D_i[s_0, \ldots, s_{t'}, \ldots, \{s_{t+1} | \pi_{t+1} = \downarrow\}]$, where for all vertices $v_{t'}$ in X_i with $\pi_{t'} \in \{\downarrow, \emptyset\}$ we have $|s_{t'}| + d_i(v_{t'}, v_{t+1}) > |s_{t+1}|$, it is $D_i[s_0, \ldots, s_{t'}, \ldots, \{s_{t+1} | \pi_{t+1} = \downarrow\}] = \infty$ (3.30). Note also that by the second property of a tree decomposition, for every edge $(u, v) \in E$ there must be a node i with $u, v \in X_i$ (introducing either u or v, for a nice tree decomposition) and an entry in the node's table where such a partial solution will be valid, if one exists.

  For entries $D_i[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \emptyset\}]$, properties (3.25-3.27) and (4.12) hold as they did for the corresponding partial solutions of the previous table. For (3.29), there may be some partial solutions corresponding to entries of the previous table D_{i-1}, in which vertices v_j ($\forall j \in [0, t']$, for some $t' \in [0, t]$) had a positive state with $\pi_j = \uparrow$ and $r - |s_j| \leq d_i(v_j, v_{t+1})$, implying that all entries $D_i[S, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \emptyset\}]$, $\forall S \in \mathcal{S}_{t'}^+$ can refer to extensions of the same partial solutions, since the "expected" connection v_{t+1} for these v_j has been introduced (and negative states are inclusive for each bag), meaning the overall minimum of these values is sufficient.

  For all entries $D_i[s_0, \ldots, s_t, \{s_{t+1} | \pi_{t+1} = \uparrow\}]$, all partial solutions for the corresponding entries $D_{i-1}[s_0, \ldots, s_t]$ extend naturally with the same values, as validity of (3.23,3.30) is not affected upon introduction of a vertex with a positive state (note the reference in (3.27) to vertices v_j with $\pi_j = \uparrow$ as applying to forgotten vertices $u \in V_i \setminus X_i$ and appropriately treated next).

- Forget node i with $X_i = X_{i-1} \setminus \{v_{t+1}\}$: For all partial solutions corresponding to entries $D_i[s_0, \ldots, s_t]$, the correct values are already computed in $D_{i-1}[s_0, \ldots, s_t, s_{t+1}]$, with the exception of those partial solutions that consider v_{t+1} for inclusion in K, where $|s_{t+1}| = 0$. Thus, for (3.25) or (3.28), the value of these partial solutions should be increased by $+1$, since $v_{t+1} \in V_i \setminus X_i$. Accordingly, for partial solutions where $\pi_{t+1} = \uparrow$, there must be a vertex v_j (still) in X_i, with $\pi_j = \uparrow$ and $|s_{t+1}| + d_i(v_j, v_{t+1}) \leq |s_j|$ (3.27), which is exactly the reason the algorithm makes use of the restricted table D_{i-1}^* in the computation of forget nodes' tables. Overall, taking the minimum of previously computed values (over all choices of state s_{t+1}) with adjustments and restrictions as described above, indeed gives the correct value $D_i[s_0, \ldots, s_t]$ for all state-configurations $(s_0, \ldots, s_t)$.

- Join node i with $X_i = X_{i-1} = X_{i-2}$: In all partial solutions corresponding to entries $D_i[s_0, \ldots, s_t]$ with negative states $\pi_j = \downarrow, \forall j \in [0, t']$ and zero/positive states

$\pi_{t'+1}, \ldots, \pi_t \in \{\emptyset, \uparrow\}$ for some $t' \in [0,t]$, vertices $v_0, \ldots, v_{t'}$ will be connected to K through either of the two children's terminal subgraphs $V_{i-1} \setminus X_{i-1}$, or $V_{i-2} \setminus X_{i-2}$. These partial solutions are "mixtures" of the partial solutions computed for both child nodes, in the corresponding entries of which, vertices are assigned a negative state on one side and a positive state on the other, involving, however, the same underlying connections in both cases. Thus considering all possible state configurations (in terms of $\uparrow, \downarrow$ directions), adding the values of complementary partial solutions from both sides and retaining the overall minimum guarantees that for every v_j with $\pi_j = \downarrow$, there will either be a vertex u in $K \subseteq (V_{i-1} \setminus X_{i-1}) \cup (V_{i-2} \setminus X_{i-2})$ such that $d_i(u, v_j) \leq |s_j|$ (3.28), or some $v_{t'}$ in X_i with $s_{t'} = (\emptyset, 0)$ and $d_i(v_j, v_{t'}) \leq |s_j|$ (3.29), if that was the case for some appropriate combination of partial solutions for $i-1$ and $i-2$. For (3.25-3.27), observe that simply taking the minimum of additions for previous partial solutions keeps vertices in $V_i \setminus X_i$ covered, i.e. $\forall u \in V_i \setminus X_i$, either $\exists v \in K \subseteq (V_{i-1} \setminus X_{i-1}) \cup (V_{i-2} \setminus X_{i-2})$ with $d_i(u, v) \leq r$, or $\exists v_j \in X_i$ with $\pi_j = \emptyset$ and $d_i(u, v_j) \leq r$, or $\exists v_j \in X_i$ with $\pi_j = \uparrow$ and $d_i(u, v_j) \leq |s_j|$, since for each entry $D_i[s_0, \ldots, s_t]$ with positive states $s_{t'+1}, \ldots, s_t$ for some $t' \in [0,t]$, all vertices $v_{t'+1}, \ldots, v_t$ are assigned the same state in all entries considered.

Theorem 46. *Given graph G, along with $k, r \in \mathbb{N}^+$ and nice tree decomposition $(\mathcal{X}, T)$ of width tw for G, there exists an algorithm to solve the decision version of the (k,r)-*Center *problem in $O(2^{tw}(2r+1)^{tw} \cdot n)$ time.*

Proof. Correctness of the dynamic programming algorithm is given above, while for the final computation at the root z of T, all entries $D_z[s_0, \ldots, s_t]$ with $\pi_j \in \{\downarrow, \emptyset\}$ $(\forall j \in [0,t])$ are considered after addition to their value of the number of vertices assigned a zero state in their state-configuration. In particular, let $\mathcal{S}_t^{\leq} \coloneqq \{(s_0, \ldots, s_j, \ldots, s_t) | \pi_j \in \{\downarrow, \emptyset\}, \forall j \in [0,t]\}$ be the set of state-configurations where no vertex is assigned a positive state and for some state-configuration S, let $R_S \coloneqq \{s_j \in S | \pi_j = \emptyset, \forall j \in [0,t]\}$ be the set of zero states. The algorithm outputs "yes", if $\min_{S \in \mathcal{S}_t^{\leq}} \{D_r[S] + |R_S|\} \leq k$ and "no" otherwise.

For the algorithm's complexity, there are $(2r+1)^{\text{tw}}$ entries for each table D_i of any node $i \in I$, with $|I| = O(\text{tw} \cdot |V|)$ for nice tree decomposition $(\mathcal{X}, T = (I, F))$, while any entry can be computed in time $O(1)$ for leaf nodes, $O(r)$ for forget nodes and $O(2^{\text{tw}})$ for join nodes (on account of $|\mathcal{S}_{t'}^-| = O(2^{\text{tw}})$). For introduce nodes, all $(2r)^{\text{tw}}$ entries for which introduced vertex v_{t+1} is assigned a non-zero state $s_{t+1} \neq (\emptyset, 0)$ require $O(1)$ time, while entries where $s_{t+1} = (\emptyset, 0)$ require at most $O(2^{\text{tw}})$ (on account of $|\mathcal{S}_{t'}^+| = O(2^{\text{tw}})$), giving an overall running-time of $O(2^{\text{tw}}(2r+1)^{\text{tw}} \cdot n)$, as claimed. □

4

On the Structurally Parameterized d-Scattered Set Problem

In this chapter we study the d-Scattered Set problem. The problem can already be seen to be hard, as it generalizes Independent Set (for $d = 2$), while an alternative name is Distance-d Independent Set [43, 81, 42]. This hardness prompts the analysis of the problem when the input graph is of restricted structure, our aim being to provide a comprehensive account of the complexity of d-Scattered Set through various upper and lower bound results. Our viewpoint is parameterized: we consider the well-known structural parameters treewidth **tw**, tree-depth **td**, vertex cover number **vc** and feedback vertex set number **fvs**, that principally express the intended restrictions on the input graph's structure, while we examine both the edge-weighted and unweighted variants.

Before we describe our results in detail, we note that they proceed along similar lines as those of the previous chapter. Indeed, our analysis of the structurally parameterized properties of the d-Scattered Set problem is analogous to that performed for (k, r)-Center and our results in fact comparable. This is partly due to the proximity of techniques used to obtain them, but can also be seen as a consequence of the similarity of the influence of distance-based generalizations on the functions of independence and domination.

Perhaps the main observation we can make on this similarity is that, apart from its converse signification for each problem (i.e. requiring distances to be either $\geq d$ or $\leq r$), the distance parameters in each case measure the extent of the influence a vertex can exert across its region in the graph and thus imply a division of the landscape into 'areas of influence', that must be efficiently arranged (i.e. *packed* or *covered*) for optimality. As an example, this is reflected in our dynamic programming algorithms (Theorem 24 and Theorem 51, see also Theorem 46 and [22, 50]) and the state-representations they employ, with an apparent correspondence between the significance of distances $d/2$ and r (i.e. multiples of an edge in both cases).[1] Justification for the necessity of such state-representations is given by the matching lower bounds (Theorem 21 and Theorem 50), meaning that our approach is not unreasonably applicable in each case, therefore also unifying the classification of both problems' structurally parameterized complexity.

Our results: First, in Subsection 4.1.1 we present a lower bound of $(d-\epsilon)^{\mathrm{tw}} \cdot n^{O(1)}$ on the complexity of any algorithm solving d-Scattered Set parameterized by treewidth tw, based on the SETH. This result can be seen as an extension of the bound of $(2-\epsilon)^{\mathrm{tw}} \cdot n^{O(1)}$

[1] This *duality* between problems and distances r and $d/2$ is also discussed in [88].

for Independent Set ([76]) for larger values of d, for which the construction is required to be much more compact in terms of encoded information per unit of treewidth. In Subsection 4.1.2 we provide a dynamic programming algorithm of running-time $O^*(d^{\text{tw}})$, matching this lower bound, over a given tree decomposition of width tw. The algorithm actually solves the counting version of d-Scattered Set, making use of standard techniques (dynamic programming on tree decompositions), with an application of the fast subset convolution technique of [9] (or *state changes* [15, 94]) to bring the running-time down to match the size of the dynamic programming tables.

Having thus identified the complexity of the problem with respect to tw, we next focus on the more general parameters vc and fvs and we show in Subsection 4.2.1 that the edge-weighted d-Scattered Set problem parameterized by vc $+ k$ is W[1]-hard. If, on the other hand, all edge-weights are set to 1, then d-Scattered Set (the unweighted variant) parameterized by fvs$+k$ is W[1]-hard. Our reductions also imply exponential lower bounds based on the ETH on the square root of the parameters, yet we do not believe these to be tight (as opposed to a single exponential lower bound), due our construction's quadratic increase of parameter size (as our focus lies on the edges).

We complement these results with a single-exponential algorithm for the unweighted variant, of running-time $O^*(3^{\text{vc}})$ for the case of even d, while for odd d the running-time is $O^*(4^{\text{vc}})$. The algorithm relies on defining a variant of Set Packing as a sub-problem that we solve via dynamic programming, with the difference in running-times, depending on the parity of d, being due to the number of possible situations for a vertex with respect to potential candidates for selection.

Next, for the unweighted variant we also show in Subsection 4.2.3 the existence of an algorithm parameterized by td of running-time $O^*(2^{O(\text{td}^2)})$, as well as a matching ETH-based lower bound. The upper bound follows from known connections between the tree-depth of a graph and its diameter, while the lower bound comes from a reduction from 3-SAT.

Finally, we turn again to tw in Section 4.3 and we present a FPT-AS of running-time $O^*((\text{tw}/\epsilon)^{O(\text{tw})})$ that finds a $d/(1+\epsilon)$-scattered set of size k, if a d-scattered set of the same size exists. The algorithm is based on a rounding technique introduced in [73] and it can be seen to outperform any exact algorithm for the problem (for large d, i.e. $d \geq O(\log n)$), even for the unweighted case and more restricted parameters (similarly to Section 3.3). Figure 4.1 shows the hierarchical relationships between parameters and Table 4.1 summarizes our results.

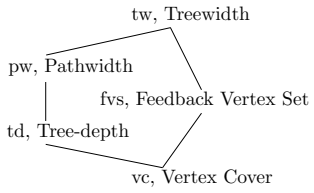


Figure 4.1: Relationships of considered parameters. Algorithmic results are inherited downwards (from cw to vc), hardness results upwards.

	tw	fvs	td	vc
FPT exact	51 (w/u)		57 (u)	56 (u)
FPT-AS	63 (w/u)			
SETH LB	50 (w/u)			
ETH LB		55 (w/u)	61 (u)	54 (w)
W[1]-hard		55 (w/u)		54 (w)

Table 4.1: A summary of our results for all considered parameters. Initials u/w denote the unweighted/weighted variants.

4.1 Treewidth

4.1.1 Lower bound based on the SETH

In this subsection we show that for any fixed $d > 2$, the existence of any algorithm for the d-Scattered Set problem of running-time $O^*((d-\epsilon)^{\text{tw}})$, for some $\epsilon > 0$, would imply the existence of an algorithm for q-SAT, of running-time $O^*((2-\delta)^n)$, for some $\delta > 0$ and any $q \geq 3$.

First, let us briefly summarize the reduction for the SETH lower bound of $(2-\epsilon)^{\text{tw}}$ for Independent Set from [76]. The reduction is based on the construction of n paths (one for each variable) on $2m$ vertices each, conceptually divided into m pairs of vertices (one for each clause), with each vertex signifying assignment of value 0 or 1 to the corresponding variable. A gadget is introduced for each clause, connected to the vertex of some path that signifies the assignment to the corresponding variable that would satisfy the clause. The pathwidth of the constructed graph (and thus also its treewidth) is (roughly) equal to the number of paths and thus a direct correspondence between a satisfying assignment and an independent set can be established, meaning an $O^*((2-\epsilon)^{\text{tw}})$-time algorithm for Independent Set would imply an $O^*((2-\epsilon)^n)$-time algorithm for SAT, for any $\epsilon > 0$.

Intuitively, the reduction for Independent Set needs to "embed" the 2^n possible variable assignments into the 2^{tw} states of some optimal dynamic program for the problem, while in our lower bound construction for d-Scattered Set we need to be able to encode these 2^n assignments by d^{tw} states and thus there can be no one-to-one correspondence between a variable and only one vertex in some bag of the tree decomposition (that the optimal dynamic program might assign states to); instead, every vertex included in some bag must carry information about the assignment for a *group* of variables. Furthermore, as now $d > 2$, in order to make the converse direction of our reduction to work, we need to make our paths sufficiently long to ensure that any solution will eventually settle into a pattern that encodes a consistent assignment, as the optimal d-scattered set may "cheat" by not selecting the same vertex from each part of some long path (periodically), a situation that would imply a different assignment for the appearances of the same variable for two different clauses (see also [22] and [76]).

Clause gadget $\hat{C}$: We first describe the construction of our clause gadget $\hat{C}$: this gadget has N *input* vertices and its purpose is to only allow for selection of one of these in any d-scattered set, along with another, standard selection. Given vertices $v_1, \ldots, v_N$, we first make N paths $A_i = (a_i^1, \ldots, a_i^{\lfloor d/2 \rfloor - 1}), \forall i \in [1, N]$ on $\lfloor d/2 \rfloor - 1$ vertices. We connect vertices a_i^1 to inputs v_i, while only for even d, we also make all vertices $a_i^{\lfloor d/2 \rfloor - 1}$ into a clique (all other endpoints of each path). We then make a path $B = (b^1, \ldots, b^{\lceil d/2 \rceil + 1})$ and we connect its endpoint $b^{\lceil d/2 \rceil + 1}$ to all $a_i^{\lfloor d/2 \rfloor - 1}$. This concludes the construction of clause gadget $\hat{C}$, while Figure 4.2 provides an illustration. Observe that any d-scattered set can only include one of the input vertices (as the distance between them is $d-1$) and the vertex b_1, being the only option at distance d from all inputs.

Construction: We will describe the construction of a graph G, given some $\epsilon < 1$ and an instance ϕ of q-SAT with n variables, m clauses and at most q variables per clause. We first choose an integer $p \geq \frac{1}{(1-\lambda)\log_2(d)}$, for $\lambda = \log_d(d-\epsilon) < 1$, for reasons that become apparent in the proof of Theorem 50. Note that for the results of this subsection, d, q

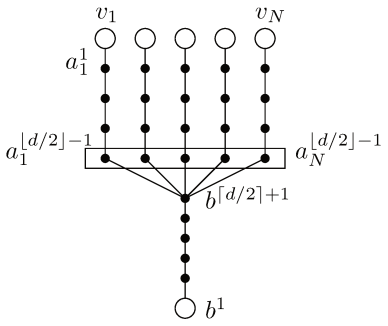


Figure 4.2: A general picture of the clause gadget $\hat{C}$ for even and odd d. Note the box indicating vertices forming a clique for the case of even d.

and p are considered constants. We then group the variables of ϕ into $t = \lceil \frac{n}{\gamma} \rceil$ groups $F_1, \ldots, F_t$, for $\gamma = \lfloor \log_2(d)^p \rfloor$, being also the maximum size of any such group.

Next, for each group F_τ of variables of ϕ, with $\tau \in [1, t]$, we make a simple gadget $\hat{G}_\tau^1$ that consists of p paths $P_\tau^l = (p_1^l, \ldots, p_d^l)$ on d vertices each, for $l \in [1, p]$. We then make $m(tp(d-1)+1)$ copies of this "column" of t gadgets $\hat{G}_1^1, \ldots, \hat{G}_t^1$, that we connect horizontally (so that we have tp "long paths"): we connect each last vertex p_d^l from a gadget $\hat{G}_\tau^j$ to vertex p_1^l from the following gadget $\hat{G}_\tau^{j+1}$, for all $l \in [1, p]$, $\tau \in [1, t]$ and $j \in [1, m(tp(d-1))]$.

Next, for every clause C_μ, with $\mu \in [1, m]$, we make $tp(d-1)+1$ copies of the clause gadget $\hat{C}^j$, for $j \in [1, m(tp(d-1)+1)]$, where for each $\mu \in [1, m]$, the number of inputs in the $tp(d-1)+1$ copies is $N = q_\mu d^p/2$, where q_μ is the number of literals in clause C_μ. One clause is assigned to each column of gadgets, so that the first m columns correspond to one clause each, with $tp(d-1)+1$ repetitions of this pattern giving the complete association. Then, for every $\tau \in [1, t]$ we associate a set $S_\tau \subset \bigcup_{l \in [1,p]} P_\tau^l$, that contains exactly one vertex from each of the p paths in $\hat{G}_\tau^j$, with an assignment to the variables in group F_τ. As there are at most $2^\gamma = 2^{\lfloor \log_2(d)^p \rfloor}$ assignments to the variables in F_τ and $d^p \geq 2^\gamma$ such sets S_τ, the association can be unique for each τ. Now, for every literal appearing in clause C_μ, exactly half of the partial assignments to the group F_τ that contains the literal's variable will satisfy it and thus, each of the $q_\mu d^p/2$ input vertices of the clause gadget will correspond to one literal and one assignment to the variables of the group that satisfy it.

Let v be an input vertex of a clause gadget $\hat{C}^j$, corresponding to a literal of clause C_μ that is satisfied by a partial assignment to the variables of group F_τ that is associated with set $S_\tau \subset \bigcup_{l \in [1,p]} P_\tau^l$, containing exactly one vertex from each path $P_\tau^l, l \in [1, p]$, from gadget $\hat{G}_\tau^j$. For even d, we then make a path $w_1, \ldots, w_{d/2-1}$ on $d/2-1$ vertices, connecting vertex w_1 to v and for each vertex $p_i^l \notin S$ of each path $P_\tau^l \in \hat{G}_\tau^j$ we also make a path $y_1, \ldots, y_{d/2-1}$ on $d/2-1$ vertices, attaching endpoint y_1 to its corresponding path vertex p_i^l, while the other endpoints $y_{d/2-1}$ are all attached to vertex $w_{d/2-1}$ and to each other (into a clique). For odd d, we make a similar construction for each such v, only the number of vertices in constructed paths is now $\lfloor d/2 \rfloor$ instead of $d/2-1$ and vertices $y_{\lfloor d/2 \rfloor}$ are not made into a clique. In this way, every input vertex v of some clause gadget is at distance exactly $d-1$ from every path vertex that does not belong to the set associated with its corresponding partial assignment (and thus exactly d from the only vertex per path that is), while the distances between any intermediate vertices via these paths are

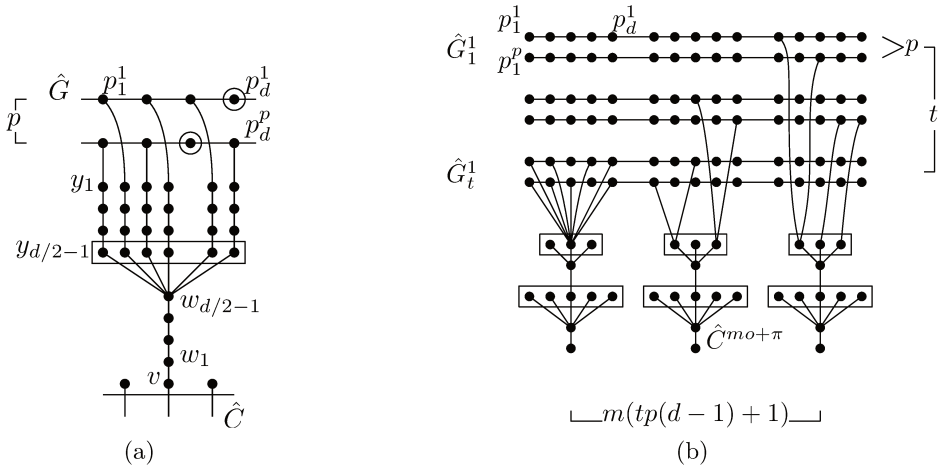


Figure 4.3: (a): The connection of an input vertex v of some clause gadget $\hat{C}$ to its corresponding path vertices in some $\hat{G}$, where vertices of set S_τ are circled and boxed vertices form a clique (for even d). (b): A simplified picture of the global construction, with some exemplative connecting paths between clause gadgets and path vertices shown as edges.

$\leq d-1$. This concludes our construction, while Figure 4.3 provides illustrations of the above.

Lemma 47. *If ϕ has a satisfying assignment, then G has a d-scattered set of size $(tp+2)m(tp(d-1)+1)$.*

Proof. Given a satisfying assignment for ϕ, we will show the existence of a d-scattered set K of G of size $|K| = (tp+2)m(tp(d-1)+1)$. Set K will include one vertex from each of the p paths in each gadget $\hat{G}_\tau^j$, with $\tau \in [1,t], j \in [1, m(tp(d-1)+1)]$ and two vertices from each clause gadget $\hat{C}^j$. In particular, for each group F_τ of variables we consider the restriction of the assignment for ϕ to these variables and identify the set S_τ associated with this partial assignment, adding all vertices of S_τ from each $\hat{G}_\tau^j$ into set K. Then, for each clause C_μ, we identify one satisfied literal (which must exist as the assignment for ϕ is satisfying) and the vertex v that corresponds to that literal and the partial assignment associated with set S_τ selected from the group of paths $P_\tau^l, l \in [1,p]$ within gadget $\hat{G}_\tau^j$, for group F_τ, in which that literal's variable appears in. We add to K every such vertex v and also vertex b_1 from each clause gadget $\hat{C}^j$, for all $j \in [1, m(tp(d-1)+1)]$, thus completing the selection and what remains is to show that K is indeed a d-scattered set of G.

To that end, observe that the pattern of our $p \cdot m(tp(d-1)+1)$ selections of all vertices from every set S_τ for each $\tau \in [1,t]$ is repeating: we have selected every d-th vertex from each "long path", since the association between sets S_τ and partial assignments for F_τ is the same of each τ. Thus on each of the tp long paths, every selected vertex is at distance exactly d both from its predecessor and its follower. Furthermore, for each clause gadget $\hat{C}^j$, with $j \in [1, m(tp(d-1)+1)]$, selected vertices v and b_1 are at distance exactly d via the gadget, while vertex v is at distance exactly d from each selected $p_i^l \in S_\tau$ from each path P_τ^l, $l \in [1,p]$, as there are only paths of length $d-1$ from v to the neighbors p_{i-1}^l, p_{i+1}^l of the selected vertex from each path. Finally, observe that the distance between vertices

on different paths P_τ^l (and thus possible selections) via the paths attached to some input vertex is always $\geq 2d-2$. □

Lemma 48. *If G has a d-scattered set of size $(tp+2)m(tp(d-1)+1)$, then ϕ has a satisfying assignment.*

Proof. Given a d-scattered set K of G of size $|K| = (tp+2)m(tp(d-1)+1)$, we will show the existence of a satisfying assignment for ϕ. First, observe that from each gadget $\hat{G}_\tau^j$, for $\tau \in [1,t], j \in [1, m(tp(d-1)+1)]$, at most p vertices can be selected, one from each path $P_\tau^l, l \in [1,p]$ within each gadget. This leaves 2 vertices to be selected from each column $j \in [1, m(tp(d-1)+1)]$ of gadgets. As the distances between some input vertex v and some path vertex p_i^l is equal to d only if the path vertex belongs to the set S_τ associated with the partial assignment to the variables of F_τ that would satisfy the literal (whose variable belongs to F_τ) corresponding to the input vertex v and $d-1$ otherwise, while the distances between any pair of input vertices are $d-1$ via the gadget with only vertex b_1 at distance exactly d from each input vertex, it is not hard to see that the only option is to select each vertex b_1 and one input vertex from each gadget $\hat{C}^j$, for $j \in [1, m(tp(d-1)+1)]$: no path vertex p_i^l could be selected with any vertex on the paths attached to some input vertex v, while no other vertex but b_1 could be selected with some input vertex of each clause gadget. Furthermore, the selection of an input vertex v must also be in agreement with each selection from the p paths to which v is connected to (via the paths of length $d-1$), i.e. the selected vertices from each path must be exactly the set S_τ that is associated with the partial assignment that satisfies the literal corresponding with v.

Next, we require that there exists at least one $o \in [0, tp(d-1)]$ for every $\tau \in [1,t]$ for which $K \cap \{\bigcup_{l\in[1,p]} P_\tau^l\}$ is the same in all gadgets $\hat{G}_\tau^{mo+\pi}$ with $\pi \in [1,m]$, i.e. that there exist m successive copies of the gadget for which the pattern of selection of vertices from paths P_τ^l does not change. As noted above, set K must contain one vertex from each such path in each gadget, while the distance between any two successive selections (on the same "long path") must be at least d. Now, depending on the starting selection, observe that the pattern can "shift towards the right" at most $d-1$ times, without affecting whether the total number of selections is exactly $m(tp(d-1)+1)$ from each "long path": the first vertex of a path can be selected within a gadget, the second vertex can be selected from its follower, the third from the one following it and so on. For each $l \in [1,p]$, this can happen at most $d-1$ times, thus at most $p(d-1)$ for each $\tau \in [1,t]$ and $tp(d-1)$ over all τ. By the pigeonhole principle, there must thus exist an $o \in [0, tp(d-1)]$ such that no such shift happens among the gadgets $\hat{G}_\tau^{mo+\pi}$, for all $\tau \in [1,t]$ and $\pi \in [1,m]$.

Our assignment for ϕ is then given by the selections for K in each gadget $\hat{G}_\tau^{mo+1}$ for this o: for every group F_τ we consider the selection of vertices from P_τ^l for $l \in [1,p]$, forming subset S_τ and its associated partial assignment to the variables of F_τ. In this way we get an assignment to all the variables of ϕ. To see why this also satisfies every clause C_π with $\pi \in [1,m]$, consider clause gadget $\hat{C}^{mo+\pi}$: there must be an input vertex v selected from this gadget, corresponding to a satisfying partial assignment for some literal of C_π, that must be at distance exactly d from each path selection that together give subset S_τ, the subset associated with this satisfying partial assignment. □

Lemma 49. *Graph G has treewidth $tw(G) \leq tp + qd^p/2 + d$.*

Proof. We will in fact show a pathwidth bound of $\text{pw}(G) \leq tp + qd^p/2 + d$ by providing a mixed strategy to clean G using $tp + qd^p/2 + d$ searchers. The claimed bound on the

treewidth then follows from lemmas 3 and 4.

We initially place one searcher on every first vertex p_1^l of every path P_τ^l in each gadget $\hat{G}_\tau^1$ for all $l \in [1,p]$ and $\tau \in [1,t]$. We also place a searcher on vertex b_1 of clause gadget $\hat{C}^1$, also one on each of its $q_\mu d^p/2$ vertices $a_i^{\lfloor d/2 \rfloor - 1}$ (between the inputs and b_1) and finally one searcher on each of the $d-1$ vertices $y_{d/2-1}$ (or $y_{\lfloor d/2 \rfloor}$ for odd d) that are connected through a $w_1, \ldots, w_{d/2-1}$ path to each input vertex (or $w_1, \ldots, w_{\lfloor d/2 \rfloor}$).

We then slide the searcher on b_1 over the path $b_1, \ldots, b^{\lceil d/2 \rceil}$ until all the path's edges as well as the edges between $b^{\lceil d/2 \rceil}$ and every $a_i^{\lfloor d/2 \rfloor - 1}$ are cleaned (the clique edges between the $a_i^{\lfloor d/2 \rfloor - 1}$ for even d are also clean). We then slide the searchers from the $a_i^{\lfloor d/2 \rfloor - 1}$ along each path to each input vertex and from there on along the paths $w_1, \ldots, w_{d/2-1}$ (or $w_{\lfloor d/2 \rfloor}$ for odd d). In this way all these paths and the edges between the $w_{d/2-1}$ and $y_{d/2-1}$ (or $w_{\lfloor d/2 \rfloor}$ and $y_{\lfloor d/2 \rfloor}$ for odd d) are cleaned and we can slide the searchers from each $y_{d/2-1}$ down to each y_1 (being adjacent to one path vertex each). We then slide all tp searchers from the first vertices p_1^l along their paths P_τ^l for $l \in [1,p]$ in each gadget $\hat{G}_\tau^1$. After all edges of the first column have been cleaned in this way, we slide the tp searchers on the first vertices of each path of the following column, we remove the searchers from the vertices of the clause gadget (and adjacent paths) and place them on their corresponding starting positions on the following column. We then repeat the above process until all columns have been cleaned. We thus use at most $tp + qd^p/2 + d$ searchers simultaneously, where $qd^p/2 + d = O(1)$. □

Theorem 50. *For any fixed $d > 2$, if d-*Scattered Set *can be solved in $O^*((d-\epsilon)^{tw(G)})$ time for some $\epsilon > 0$, then there exists some $\delta > 0$, such that q-*SAT *can be solved in $O^*((2-\delta)^n)$ time, for any $q \geq 3$.*

Proof. Assuming the existence of some algorithm of running-time $O^*((d-\epsilon)^{\mathrm{tw}(G)}) = O^*(d^{\lambda \mathrm{tw}(G)})$ for d-Scattered Set, where $\lambda = \log_d(d-\epsilon)$, we construct an instance of d-Scattered Set given a formula ϕ of q-SAT, using the above construction and then solve the problem using the $O^*((d-\epsilon)^{\mathrm{tw}(G)})$-time algorithm. Correctness is given by Lemma 47 and Lemma 48, while Lemma 49 gives the upper bound on the running-time:

$$
\begin{aligned}
O^*(d^{\lambda \mathrm{tw}(G)}) &\leq O^*\left(d^{\lambda(tp+f(d,\epsilon,q))}\right) && (4.1)\\
&\leq O^*\left(d^{\lambda p\left\lceil \frac{n}{\lfloor \log_2(d)^p \rfloor}\right\rceil}\right) && (4.2)\\
&\leq O^*\left(d^{\lambda p\frac{n}{\lfloor \log_2(d)^p \rfloor}+\lambda p}\right) && (4.3)\\
&\leq O^*\left(d^{\lambda\frac{np}{\lfloor p\log_2(d) \rfloor}}\right) && (4.4)\\
&\leq O^*\left(d^{\delta'\frac{n}{\log_2(d)}}\right) && (4.5)\\
&\leq O^*(2^{\delta'' n}) = O((2-\delta)^n) && (4.6)
\end{aligned}
$$

for some $\delta, \delta', \delta'' < 1$. Observe that in line (4.2) the function $f(d, \epsilon, q) = qd^p/2 + d$ is considered constant, as is λp in line (4.4), while in line (4.5) we used the fact that there always exists a $\delta' < 1$ such that $\lambda \frac{p}{\lfloor p \log_2(d) \rfloor} = \frac{\delta'}{\log_2(d)}$, as we have:

$$
\begin{aligned}
p \log_2(d) - 1 &< \lfloor p \log_2(d) \rfloor \\
\Leftrightarrow \frac{\lambda p \log_2(d)}{p \log_2(d) - 1} &> \frac{\lambda p \log_2(d)}{\lfloor p \log_2(d) \rfloor}, \\
\text{from which, by substitution, we get } \frac{\lambda p \log_2(d)}{p \log_2(d) - 1} &> \delta', \\
\text{now requiring } \frac{\lambda p \log_2(d)}{p \log_2(d) - 1} &\leq 1, \\
\text{or } p &\geq \frac{1}{(1 - \lambda) \log_2(d)},
\end{aligned}
$$

that is precisely our definition of p. This concludes the proof. □

4.1.2 Dynamic Programming algorithm

In this subsection we present an $O^*(d^{\text{tw}})$-time dynamic programming algorithm for the counting version of the d-Scattered Set problem. The input is a graph $G = (V, E)$, a nice tree decomposition $(\mathcal{X}, T)$ for G, where $T = (I, F)$ is a tree and $\mathcal{X} = \{X_i | i \in I\}$ is the set of bags, while $\max_{i \in i} |X_i| - 1 = \text{tw}$, along with two numbers $k \in \mathbb{N}^+, d \geq 2$, while the output is the number of d-scattered sets of size k in G.

Table description: There is a table D_i associated with every node $i \in I$ of the tree decomposition with $X_i = \{v_0, \ldots, v_t\}, 0 \leq t \leq \text{tw}$, while each table entry $D_i[\kappa, s_0, \ldots, s_t]$ contains the number of (disjoint) d-scattered sets $K \subseteq V_i$ of size $|K| = \kappa$ (its *partial solution*) and is indexed by a number $\kappa \in [1, k]$ and a $t+1$-sized tuple $(s_0, \ldots, s_t)$ of *state-configurations*, assigning a state $s_j \in [0, d-1]$ to each vertex $v_j, \forall j \in [0, t]$. There are d possible states for each vertex, designating its distance to the closest selection at the "current" stage of the algorithm:

- *Zero* state $s_j = 0$ signifies vertex v_j is considered for selection in the d-scattered set and is at distance at least d from any other such selection: $\forall u \in K : d(u, v_j) \geq d$.
- *Low* states $s_j \in [1, \lfloor d/2 \rfloor]$ signify vertex v_j is at distance at least s_j from its closest selection and at least $d - s_j$ from the second closest: $\forall u, w \in K | d(u, v_j) \leq d(w, v_j) : d(u, v_j) \geq s_j \wedge d(w, v_j) \geq d - s_j$.
- *High* states $s_j \in [\lfloor d/2 \rfloor + 1, d-1]$ signify vertex v_j is at distance at least s_j from its closest selection: $\forall u \in K : d(u, v_j) \geq s_j$.

For a node $i \in I$, each table entry $D_i[\kappa, s_0, \ldots, s_t]$ contains the number of d-scattered sets $K \subseteq V_i$ of the terminal subgraph G_i, such that the situation of each vertex in the corresponding bag is being described by the particular state configuration $(s_0, \ldots, s_t)$ indexing this entry. The computation of each entry is based on the type of node the table is associated with (leaf, introduce, forget, or join), previously computed entries of the table associated with the preceding node(s) and the structure of the node's terminal subgraph. In particular, we have $\forall i \in I, D_i[\kappa, s_0, \ldots, s_t] : [1, k] \times [0, d-1]^{t+1} \mapsto \mathbb{N}^0$, where $0 \leq t \leq \text{tw}$. The inductive computation of all table entries for each type of node follows.

Leaf node i with $X_i = \{v_0\}$:

$$D_i[\kappa, s_0] \coloneqq \begin{cases} 1 & \text{, if } s_0 = 0, \kappa = 1; \\ 1 & \text{, if } s_0 > 0, \kappa = 0; \\ 0 & \text{, otherwise.} \end{cases}$$

Leaf nodes contain only one vertex and there is one d-scattered set that includes this vertex for ($s_0 = 0$, $\kappa = 1$) and one d-scattered set that does not (for $s_0 > 0$, $\kappa = 0$).

Introduce node i with $X_i = X_{i-1} \cup \{v_{t+1}\}$:

$$D_i[\kappa, s_0, \ldots, s_t, s_{t+1}] \coloneqq \begin{cases} D_{i-1}[\kappa, s_0, \ldots, s_t], & \text{if } s_{t+1} \in [1, d-1] \text{ and} \\ & s_{t+1} \leq \min_{v_j \in X_{i-1}}(d(v_{t+1}, v_j) + s_j); \\ D_{i-1}[\kappa', s'_0, \ldots, s'_t], & \text{if } s_{t+1} = 0, \kappa' = \kappa - 1 \text{ and } \forall v_j \in X_{i-1} \text{ with } s_j = 0, \\ & \text{it is } d(v_{t+1}, v_j) \geq d, \text{ and} \\ & \forall v_j \in X_{i-1} \text{ with } d(v_{t+1}, v_j) \leq \lfloor d/2 \rfloor, \\ & \text{it is } s_j \leq d(v_{t+1}, v_j) \text{ and } s'_j = d - s_j, \\ & \text{with } s'_j = s_j, \text{ if } d(v_{t+1}, v_j) > \lfloor d/2 \rfloor; \\ 0, & \text{otherwise.} \end{cases}$$

When a vertex is introduced, for previously computed partial solutions to be correctly extended, we require that its given state matches the distance/state conditions of the other vertices in the bag, while if the introduced vertex is considered for selection, then the previous entries we examine must ensure this selection is possible.

Forget node i with $X_i = X_{i-1} \setminus \{v_{t+1}\}$:

$$D_i[\kappa, s_0, \ldots, s_t] \coloneqq \sum_{s_{t+1} \in [0, d-1]} \{D_{i-1}[\kappa, s_0, \ldots, s_t, s_{t+1}]\}.$$

The correct value for each entry is the sum over all states of the forgotten vertex v_j, where the size of the d-scattered sets is κ.

Join node i with $X_i = X_{i-1} = X_{i-2}$: Given state-tuple $(s_0, \ldots, s_t)$, we assume (without loss of generality) that for some $t' \in [0, t]$ (if any) it is $s_j \in [1, \lfloor d/2 \rfloor], \forall j \in [0, t']$ and also $\min_{\forall v_l \in X_i | s_l = 0} d(v_j, v_l) > \lfloor d/2 \rfloor$, while all other vertices are $v_{t'+1}, \ldots, v_t$. Now, let $\mathcal{S}^{\leq}_{t'}$ denote the set of all possible tuples $S = (s^{\leq}_0, \ldots, s^{\leq}_{t'})$, where each state $s^{\leq}_j$ is either the same state s_j, or its symmetrical (around $d/2$):

$$\mathcal{S}^{\leq}_{t'} \coloneqq \{(s^{\leq}_0, \ldots, s^{\leq}_{t'}) | \forall j \in [0, t'] : \{(s^{\leq}_j = s_j) \vee (s^{\leq}_j = d - s_j)\}\},$$

while for some tuple $S \in \mathcal{S}^{\leq}_{t'}$, let $\bar{S}$ denote the complementary tuple (where the state of each vertex is likewise reversed) and also let κ'' denote the number of zero states in $(s_{t'+1}, \ldots, s_t)$. Then we have:

$$D_i[\kappa, s_0, \ldots, s_t] \coloneqq \sum_{S \in \mathcal{S}^{\leq}_{t'}} \{D_{i-1}[\kappa', S, s_{t'+1}, \ldots, s_t] \cdot D_{i-2}[\kappa - \kappa' + \kappa'', \bar{S}, s_{t'+1}, \ldots, s_t]\}.$$

For join nodes, the bags of both children contain the same set of vertices, yet the partial solutions characterized by the entries of each table concern distinct terminal subgraphs G_{i-1} and G_{i-2}. For state-configurations where some vertices are of low state (that is not justified by the presence of some vertex of zero state within the bag), the closest selection to these vertices (that gives the state) might be in any of the two terminal subgraphs, but not both: if the "target" state is $s_j \in [1, \lfloor d/2 \rfloor]$, then there might be a selection in G_{i-1} at distance s_j but there must not be another selection in G_{i-2} at distance $\leq d - s_j$ (and vice-versa).

State changes: The computations at a join node as described above would add an additional factor in the complexity of our algorithm if implemented directly, yet this can be avoided by an application of the *state changing* technique (or fast subset convolution, see [9, 15, 94] and Chapter 11 from [35]): since the number of entries involved can be exponential in tw (due to the size of $\mathcal{S}^{\leq}_{t'}$), in order to efficiently compute the table for a join node i, we will first transform the tables D_{i-1}, D_{i-2} of its children into tables D^*_{i-1}, D^*_{i-2} of a new type that employs a different state representation, for which the join operation can be efficiently performed to produce table D^*_i, that we will finally transform back to table D_i, thus progressing with our dynamic programming algorithm.

In particular, each entry $D^*_i[\kappa, s_1, \ldots, s_t]$ of the new table will be an aggregate of entries $D_i[\kappa, s_0, \ldots, s_t]$ of the original table, with its value equal to the sum of the appropriate values of the corresponding entries. For vertex v_j, each low state $s_j \in [1, \lfloor d/2 \rfloor]$ in the new state signification for table D^*_i that is not justified by the presence of an appropriate selection within the bag (i.e. its minimum distance to any zero-state vertex is at least $\lfloor d/2 \rfloor + 1$) will correspond to both the same low state s_j and its symmetrical high state $d - s_j$ from the original signification. Observe that these correspondences exactly parallel the definition of set $\mathcal{S}^{\leq}_{t'}$ used in the original computations.

First, let D^*_i be a copy of table D_i. The transformation then works in t steps, vertex-wise: we require that all entries $D^*_i[\kappa, s^*_0, \ldots, s^*_t]$ contain the sum of all entries of D_i where for low states s^*_j (that are also not justified by some present selection), it is $s^*_j = s_j$ or $s^*_j = d - s_j$, and all other vertex-states and κ are fixed: at step j, we add $D^*_i[\kappa, s_0, \ldots, s_j, \ldots, s_t] = D_i[\kappa, s_0, \ldots, s_j, \ldots, s_t] + D_i[\kappa, s_0, \ldots, d - s_j, \ldots, s_t]$ if $s_j \in [1, \lfloor d/2 \rfloor]$ and $\min_{\forall v_l \in X_i | s_i = 0} d(v_j, v_l) > \lfloor d/2 \rfloor$. We then proceed to the next step for v_{j+1} until table D^*_i is computed. Observe that the above procedure is fully reversible:[2] to invert table D^*_i back to table D_i, we again work in t steps, vertex-wise: we first let D_i be a copy of D^*_i and then at step j for all other vertex-states and κ fixed, we subtract $D_i[\kappa, s_0, \ldots, s_j, \ldots, s_t] = D^*_i[\kappa, s_0, \ldots, s_j, \ldots, s_t] - D^*_i[\kappa, s_0, \ldots, d - s_j, \ldots, s_t]$ if $s_j \in [1, \lfloor d/2 \rfloor]$ and $\min_{\forall v_l \in X_i | s_i = 0} d(v_j, v_l) > \lfloor d/2 \rfloor$. For both transformations, we perform at most one addition per $k \cdot d^{t+1}/2$ entries for each step $j \in [0, t]$.

Thus we can compute table D^*_i by simply multiplying the values of the two corresponding entries from D^*_{i-1}, D^*_{i-2}, as they now contain all required information for this state representation, with the inverse transformation of the result giving table D_i:

$$D^*_i[\kappa, s_0, \ldots, s_t] \coloneqq \sum_{\kappa'=0}^{\kappa'=\kappa} D^*_{i-1}[\kappa', s_0, \ldots, s_t] \cdot D^*_{i-2}[\kappa - \kappa' + \kappa'', s_0, \ldots, s_t].$$

[2]This is the reason for *counting* the number of solutions for each κ: there is no additive inverse operation for the max-sum semiring, yet the sum-product ring is indeed equipped with subtraction.

Theorem 51. *Given a graph G, along with $d \in \mathbb{N}^+$ and nice tree decomposition $(\mathcal{X}, T)$ of width tw for G, there exists an algorithm to solve the counting version of the d-*Scattered Set *problem in $O^*(d^{tw})$ time.*

Proof. Let $U_i(\kappa, s_0, \ldots, s_t) = \{K \subseteq V_i | K \cap X_i = \{v_j \in X_i | s_j = 0\}, |K| = \kappa, \forall u, v \in K : d(u, v) \geq d\}$ be the set of all d-scattered sets in G_i of size κ. To show correctness of our algorithm we need to establish that for every node $i \in I$, each table entry $D_i[\kappa, s_0, \ldots, s_t]$ contains the size of a partial solution to the problem as restricted to G_i, i.e. the size of this set $|U_i(\kappa, s_0, \ldots, s_t)|$, such that the distance between every pair of vertices in K is at least d, while for every vertex $v_j \in X_i$, its state for this entry describes its situation within this partial solution. In particular, we need to show the following:

$$\forall i \in I, \forall \kappa \in [1, k], \forall (s_0, \ldots, s_t) \in [0, d-1]^{t+1}, 0 \leq t < \text{tw} : \tag{4.7}$$
$$D_i[\kappa, s_0, \ldots, s_t] = |U_i(\kappa, s_0, \ldots, s_t)| : \tag{4.8}$$
$$\{\forall u, w \in K : d(u, w) \geq d\} \wedge \tag{4.9}$$
$$\wedge \{\forall v_j \in X_i | s_j \in [1, \lfloor d/2 \rfloor], \forall u, w \in K | d(u, v_j) \leq d(w, v_j) : \tag{4.10}$$
$$d(u, v_j) \geq s_j \wedge d(w, v_j) \geq d - s_j\} \wedge \tag{4.11}$$
$$\wedge \{\forall v_j \in X_i | s_j \in [\lfloor d/2 \rfloor + 1, d-1], \forall u \in K : d(u, v_j) \geq s_j\}. \tag{4.12}$$

In words, the above states that for every node i, $\kappa \in [1, k]$ and all possible state-configurations $(s_0, \ldots, s_t)$ (4.7), table entry $D_i[\kappa, s_0, \ldots, s_t]$ contains the size of set $U_i(\kappa, s_0, \ldots, s_t)$ containing all subsets K of V_i (that include all vertices $v_j \in X_i$ of state $s_j = 0$) of size $|K| = \kappa$ (4.8), such that the distance between every pair of vertices $u, w \in K$ is at least d (4.9), for every vertex $v_j \in X_i$ with low state $s_j \in [1, \lfloor d/2 \rfloor]$ and a pair of vertices u, w from K with u closer to v_j than w (4.10), its distance to u is at least equal to its state s_j, while its distance to w is at least $d - s_j$ (4.11), while for every vertex $v_j \in X_i$ with high state $s_j \in [\lfloor d/2 \rfloor + 1, d-1]$ and a vertex u from K, its state s_j is at most its distance to u (4.12). This is shown by induction on the nodes $i \in I$:

- Leaf node i with $X_i = \{v_0\}$: This is the base case of our induction. There is only one d-scattered set K in V_i of size $\kappa = 1$, for which (4.9-4.12) is true, that includes v_0 and only one for $\kappa = 0$ that does not. In the following cases, we assume (our inductive hypothesis) that all entries of D_{i-1} (and D_{i-2} for join nodes) contain the correct number of sets K.

- Introduce node i with $X_i = X_{i-1} \cup \{v_{t+1}\}$: For entries with $s_j \in [1, \lfloor d/2 \rfloor]$, validity of (4.9,4.12) is not affected, while for (4.10-4.11): it is $s_{t+1} \leq d(v_{t+1}, v_j) + s_j$ for some vertex $v_j \in X_{i-1}$, for which, by the induction hypothesis we have that $s_j \leq d(u, v_j)$ and $d - s_j \leq d(w, v_j)$, where u is the closest selection to v_j and w the second closest. To see the same holds for v_{t+1}, observe that $s_{t+1} \leq d(v_{t+1}, v_j) + d(u, v_j) \leq d(v_{t+1}, u)$ (by substitution) and $d - s_j \leq d(w, v_j) \Rightarrow d - d(w, v_j) \leq s_j \Rightarrow d - d(w, v_j) + d(v_j, v_{t+1}) \leq d(v_{t+1}, v_j) + s_j \Rightarrow d - d(w, v_{t+1}) \leq s_{t+1} \Rightarrow d - s_{t+1} \leq d(w, v_{t+1})$.

  For entries with $s_{t+1} \in [\lfloor d/2 \rfloor + 1, d-1]$, validity of (4.9-4.11) is not affected, while for (4.12): it is $s_{t+1} \leq d(v_{t+1}, v_j) + s_j$ for some vertex $v_j \in X_{i-1}$, for which, by the induction hypothesis we have that $s_j \leq d(u, v_j)$ and thus $s_{t+1} \leq d(v_{t+1}, v_j) + d(u, v_j) \leq d(v_{t+1}, u)$.

For entries with $s_{t+1} = 0$, observe that the low states s_j of vertices $v_j \in X_{i-1}$ in the new entry with $d(v_{t+1}, v_j) \leq \lfloor d/2 \rfloor$ (for otherwise their situation has not changed by addition of v_{t+1} with $s_{t+1} = 0$) would correspond to a high original state $s'_j = d - s_j$ in the previously computed entry, for which partial solution we know that $d(v_j, u) \geq s'_j, \forall u \in K$, or that the previously closest selection was at distance at least s'_j (4.10-4.11). For high states s_j of vertices $v_j \in X_{i-1}$, the requirement is exactly $d(v_{t+1}, v_j) \geq s_j$ (4.12) and finally, for (4.9), if there was some $u \in K$ such that $u \notin X_{i-1}$ and $d(u, v_{t+1}) < d$, then there must be some $v_j \in X_{i-1}$ (on the path between u and v_{t+1}), for which if $d(v_{t+1}, v_j) \leq \lfloor d/2 \rfloor$ and s_j is low then (4.12) was false (as s'_j must have been high and matching $d(u, v_j)$), while if $d(v_{t+1}, v_j) > \lfloor d/2 \rfloor$ and s_j is high, then (4.10-4.11) was false (in all other cases it would not be $d(u, v_{t+1}) < d$).

- Forget node i with $X_i = X_{i-1} \setminus \{v_{t+1}\}$: In a forget node, the only difference for the partial solutions in which the forgotten vertex was of state $s_{t+1} = 0$ is that now vertex v_{t+1} is included in set K only and not X_i. Thus, due to (4.9), the correct number is indeed the sum over all states for v_{t+1}.

- Join node i with $X_i = X_{i-1} = X_{i-2}$: Observe that for (4.9), if there was a pair $u, w \in K \cap X_i$ or $u \in K \cap X_i, w \in K \setminus X_i$ at $d(u, w) < d$, then (4.9) was not true for either $i-1$ or $i-2$, while if there was a pair $u \in K \cap V_{i-1} \setminus X_{i-1}, w \in K \cap V_{i-2} \setminus X_{i-2}$ with $d(u, w) < d$, then there must be some vertex $v_j \in X_i$ (on the path between the two) for which (4.11) was not true. For (4.10-4.12), observe that for vertices v_j of low state s_j, (4.10-4.11) must have been true for either $i-1$ or $i-2$ and (4.12) for the other, while for vertices v_j of high state s_j it suffices that (4.12) must have been true for both.

For the algorithm's complexity, there are $k \cdot d^{\text{tw}}$ entries for each table D_i of any node $i \in I$, with $|I| = O(\text{tw} \cdot |V|)$ for nice tree decomposition $(\mathcal{X}, T = (I, F))$, while any entry can be computed in time $O(1)$ for leaf and introduce nodes, $O(d)$ for forget nodes, while the state changes can be computed in $O(k \cdot \text{tw} \cdot d^{\text{tw}})$ time, with each entry of the transformed table D_i^* computed in $O(k)$ time. □

4.2 Vertex Cover, Feedback Vertex Set and Tree-depth

4.2.1 Vertex Cover, Feedback Vertex Set: W[1]-Hardness

In this subsection we show that the edge-weighted variant of the d-Scattered Set problem parameterized by $\text{vc} + k$ is W[1]-hard via a reduction from k-Multicolored independent Set. Given an instance $[G = (V, E), k]$ of the latter, we construct an instance $[G' = (V', E'), k']$ of edge-weighted d-Scattered Set where $d = 6n$.

Construction: First, for every set $V_i \subseteq V$ we create a set $P_i \subseteq V'$ of n vertices $p_l^i, \forall l \in [1, n], \forall i \in [1, k]$ (that directly correspond to the vertices of V_i). Next, for each $i \in [1, k]$ we make a pair of vertices a_i, b_i, connecting a_i to each vertex p_l^i by an edge of weight $n + l$, while b_i is connected to each vertex p_l^i by an edge of weight $2n - l$. Next, for every non-edge $e \in \bar{E}$ (i.e. $\bar{E}$ contains all pairs of vertices from V that are not connected by an edge from E) between two vertices from different V_{i_1}, V_{i_2} (with $i_1 \neq i_2$), we make a vertex u_e that we connect to vertices a_{i_1}, b_{i_1} and a_{i_2}, b_{i_2}. We set the weights of these edges as

follows: suppose that e is a non-edge between the j_1-th vertex of V_{i_1} and the j_2-th vertex of V_{i_2}. We then set $w(u_e, a_{i_1}) = 5n - j_1$, $w(u_e, b_{i_1}) = 4n + j_1$ and $w(u_e, a_{i_2}) = 5n - j_2$, $w(u_e, b_{i_2}) = 4n + j_2$. Next, for every pair of i_1, i_2 we make two vertices g_{i_1,i_2}, g'_{i_1,i_2}. We connect g_{i_1,i_2} to all vertices u_e that correspond to non-edges e between vertices of the same pair V_{i_1}, V_{i_2} by edges of weight $(6n-1)/2$ and also g_{i_1,i_2} to g'_{i_1,i_2} by an edge of weight $(6n+1)/2$. This concludes the construction of G', with Figure 4.4 providing an illustration.

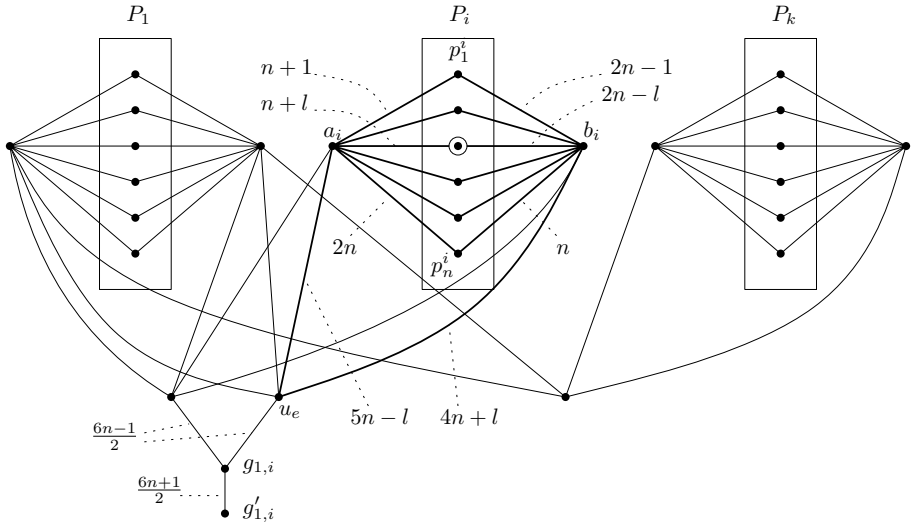


Figure 4.4: A general picture of graph G', where the circled vertex is p_l^i and dotted lines match weights to edges.

Lemma 52. *If G has a k-multicolored independent set, then G' has a $6n$-scattered set of size $k + 2\binom{k}{2} = k + k(k-1) = k^2$.*

Proof. Let $I \subseteq V$ be a multicolored independent set in G of size k and $v_{l_i}^i$ denote the vertex selected from each V_i, or $I \coloneqq \{v_{l_1}^1, \ldots, v_{l_k}^k\}$. Let $S \subseteq V'$ include the set of vertices $p_{l_i}^i$ in G' that correspond to each $v_{l_i}^i$. For any pair $i, j \in [1, k]$ of indices with $i \neq j$, let u_e be the vertex corresponding to the non-edge between vertices $v_{l_i}^i, v_{l_j}^j \in I$. All these u_e vertices exist, as I is a k-multicolored independent set. We include all these u_e vertices in S and also every $g'_{i,j}$ that is connected to $g_{i,j}$ that each such u_e is connected to. Now S is of size $k + 2\binom{k}{2}$ and we claim it is a $6n$-scattered set: all selected vertices $p_{l_i}^i$ are at distance $n + l_i + 5n - l_i = 6n$ via a_i and distance $2n - l_i + 4n + l_i = 6n$ via b_i from any selected vertex u_e that corresponds to a non-edge "adjacent" to their corresponding $v_{l_i}^i$, while every selected vertex u_e corresponding to a non-edge between two vertices of groups V_{i_1}, V_{i_2} is at distance $(6n-1)/2 + (6n+1)/2 = 6n$ from every selected vertex g'_{i_1,i_2} that is connected to g_{i_1,i_2} that connects all such u_e vertices between these groups. □

Lemma 53. *If G' has a $6n$-scattered set of size $k + 2\binom{k}{2} = k + k(k-1) = k^2$, then G has a k-multicolored independent set.*

Proof. Let $S \subseteq V'$ be the $6n$-scattered set, with $|S| = k + 2\binom{k}{2}$. As the distance via $g_{i,j}$ between any two vertices u_e, u_h corresponding to non-edges between vertices of the same groups V_i, V_j is $6n - 1$, set S can contain at most one such vertex for every such

pair of groups, their number being $\binom{k}{2}$. Since the size of S is $k + 2\binom{k}{2}$, the set must also contain one other vertex per group, the only choices available being vertices $g'_{i,j}$ at distance $(6n-1)/2 + (6n+1)/2 = 6n$ from any such u_e, leaving the k choices for at most one vertex from each P_i, as the distance between any pair $p^i_{l_1}, p^i_{l_2}$ is $2n + l_1 + l_2 < 6n$ via a_i and $4n - l_1 - l_2 < 6n$ via b_i. Now, let $u_e \in S$ be a selected vertex corresponding to a non-edge between vertices $v^i_{l_i}, v^j_{l_j}$ from groups V_i, V_j and $p^i_{o_i}, p^j_{o_j} \in S$ be the vertices selected from P_i, P_j. Vertex u_e is at distance $5n - l_i + n + o_i = 6n + o_i - l_i$ via a_i and $4n+l_i+2n-o_i = 6n+l_i-o_i$ via b_i from $p^i_{o_i} \in P_i$, while at distance $5n-l_j+n+o_j = 6n+o_j-l_j$ via a_j and $4n+l_j+2n-o_j = 6n+l_j-o_j$ via b_j from $p^j_{o_j} \in P_j$. It is not hard to see that if $o_i \neq l_i$ then u_e and $p^i_{l_i}$ cannot be together in S, while also if $o_j \neq l_j$ then u_e and $p^j_{l_j}$ cannot be together in S. Thus, there must be no edge between every pair of vertices $v^i_{l_i}, v^j_{l_j}$ that correspond to $p^i_{l_i}, p^j_{l_j} \in S$, meaning the set I that includes all such $v^i_{l_i}$ is a k-multicolored independent set. □

Theorem 54. *The edge-weighted d-*Scattered Set *problem is W[1]-hard parameterized by $vc + k$. Furthermore, if there is an algorithm for edge-weighted d-*Scattered Set *running in time $n^{o(\sqrt{vc}+\sqrt{k})}$ then the ETH is false.*

Proof. Observe that the set $Q \subset V'$ that includes all vertices $a_i, b_i, \forall i \in [1, k]$ and all vertices $g_{i,j}, \forall i \neq j \in [1, k]$ is a minimum vertex cover of G', as all edges have exactly one endpoint in Q. This means $\mathrm{vc}(G') \leq 2k + \binom{k}{2} = O(k^2)$. In addition, the relationship between the sizes of the solutions of d-Scattered Set and k-Multicolored Independent Set is $k' = |S| = k + 2\binom{k}{2} = O(k^2)$. Thus, the construction along with lemmas 52 and 53, indeed imply the statement. □

Using essentially the same reduction (with minor modifications) we also obtain similar hardness results for unweighted d-Scattered Set parameterized by fvs:

Corollary 55. *The unweighted d-*Scattered Set *problem is W[1]-hard parameterized by $fvs + k$. Furthermore, if there is an algorithm for unweighted d-*Scattered Set *running in time $n^{o(fvs+\sqrt{k})}$ then the ETH is false.*

Proof. The modifications to the above construction that we require are the following: each edge e of weight $w(e)$ is substituted by a path of length $w(e)$, apart from the edge between every g_{i_1,i_2} to every g'_{i_1,i_2} that is now a path of length $d-1 = 6n-1$ and all edges between every g_{i_1,i_2} to all adjacent u_e that correspond to non-edges between vertices of pair V_{i_1}, V_{i_2} that are now only a single edge. In this way, Lemma 52 goes through unchanged, while for Lemma 53, it suffices to observe that no two vertices anywhere on the paths between some g_{i_1,i_2} and some a_i, b_i could be selected instead of the intended selection of g'_{i_1,i_2} and some u_e that matches the selections from V_{i_1}, V_{i_2}, as the distance between any two vertices between g_{i_1,i_2} and some a_i, b_i is always $< 2d = 12n$, while if the selected vertices are not exactly some g'_{i_1,i_2} and (the correct) u_e, then the minimum distance between these selections and the closest selection from V_{i_1}, V_{i_2} will be less than d.

It is not hard to see that the set Q containing all $a_i, b_i, \forall i \in [1, k]$ is a feedback vertex set of G', as removal of all these vertices results in an acyclic graph, hence $\mathrm{fvs}(G') = O(k)$. □

4.2.2 Vertex Cover: FPT algorithm

We next show that unweighted d-Scattered Set admits an FPT algorithm, in contrast to its weighted version (Theorem 54). Given graph G and $d \geq 3$, our algorithm will first define an instance of a Set Packing variant, where elements may be *partially* included in some sets and then solve the problem by dynamic programming. In particular, we define Partial Set Packing, in which any element has a *coefficient* of inclusion in each subset, while a collection of subsets will be a solution if there is no pair of subsets for which the sum of some element's coefficients is > 1.

Our instance will then be the following: the universe will consist of the vertices of the given vertex cover and there will be a set for each vertex (roughly), with set inclusion based on whether the distance of each of the vertices belonging to the vertex cover is smaller (complete inclusion), equal (partial inclusion), or greater than $d/2$ (no inclusion). A collection of subsets will thus be a solution if no two vertices are selected for which there exists some third vertex, for which the sum of its distances to the selected pair of vertices is $< d$. Depending on the parity of d, there may be 3 or 4 options for inclusion of each vertex in some subset and thus the running-time of our algorithm is $O^*(3^{\text{vc}})$ for the case of even d and $O^*(4^{\text{vc}})$ for odd d.

Theorem 56. *Given graph G, along with $d > 2$ and a vertex cover of size vc of G, there exists an algorithm solving the unweighted d-Scattered Set problem in $O^*(3^{vc})$ time for even d and $O^*(4^{vc})$ time for odd d.*

Proof. Let C be the given vertex cover of G and $I = V \setminus C$ be the remaining independent set. Let also $Y \subseteq I$ be the subset of vertices from I with a unique neighborhood in C, i.e. for two vertices $u, v \in I$ with $N(u) = N(v)$, set Y only contains one of them. Observe that the size of Y is thus exponentially bounded by the size of C: $|Y| \leq 2^{|C|}$.

For an instance of our Partial Set Packing variant, let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be the universe of elements and $\mathcal{S} = \{S_1, \ldots, S_m\}$ be the set family. Further, for even d, we introduce a weight function $w(u_i, S_j) : \mathcal{U} \times \mathcal{S} \mapsto \{0, 1/2, 1\}$, giving the "coefficient" of element u_i for inclusion in set S_j, where 0 implies the element is not included in the set, $1/2$ implies *partial* and 1 *complete* inclusion. For odd d, the weight function $w(u_i, S_j) : \mathcal{U} \times \mathcal{S} \mapsto \{0, 1/3, 2/3, 1\}$ allows more values for partial inclusion. In our solutions to this variant we will allow any number of sets to partially include any element, yet if any set in the solution completely includes some element, then no other set that includes the same element either partially, or completely, can also be part of the same solution, i.e. a collection of subsets $S \subseteq \mathcal{S}$ will be a solution, if for every element $u_i \in \mathcal{U}$, the sum for any two pairs is at most 1: $\max_{S_a, S_b \in S}\{w(u_i, S_a) + w(u_i, S_b)\} \leq 1$.

We then define our Partial Set Packing instance as follows: we make an element $u_i \in \mathcal{U}$ for every vertex of C and a set $S_j \in \mathcal{S}$ for every vertex of $C \cup Y$. We thus have $|C|$ elements and $|C| + |Y| \leq |C| + 2^{|C|}$ sets. For even d, an element u_i with corresponding vertex $v \in C$ is included in some set S_j with corresponding vertex z completely (or $w(u_i, S_j) = 1$) if $d(v, z) < d/2$, while an element u_i with corresponding vertex $v \in C$ is included in some set S_j with corresponding vertex z partially (or $w(u_i, S_j) = 1/2$), if $d(v, z) = d/2$. For odd d, an element u_i with corresponding vertex $v \in C$ is included in some set S_j with corresponding vertex z completely (or $w(u_i, S_j) = 1$) if $d(v, z) < \lfloor d/2 \rfloor$, 2/3-partially ($w(u_i, S_j) = 2/3$) if $d(v, z) = \lfloor d/2 \rfloor$ and 1/3-partially ($w(u_i, S_j) = 1/3$) if $d(v, z) = \lceil d/2 \rceil$.

In the classic dynamic programming procedure for Set Packing we store a table $OPT[U, j]$ that contains, for every subset of elements $U \subseteq \mathcal{U}$ and $j \in [1, m]$ the maximum number of subsets that can be selected from $\{S_1, \ldots, S_j\}$, such that no element of U is included in any of them. The dynamic programming procedure then first computes for $j = 1$: $OPT[U, 1] \coloneqq 1$, if $U \cap S_j = \emptyset$ and 0 otherwise, while for $j = 2, \ldots, m$ it is: $OPT[U, j+1] \coloneqq \max\{OPT[U, j], OPT[U \cup S_{j+1}, j] + 1\}$ if $S_{j+1} \cap U = \emptyset$ and only $OPT[U, j+1] \coloneqq OPT[U, j]$ otherwise.

We will create a similar table $OPT[U, j]$ for every $j \in [1, m]$ and every $U = \{(u_i \in \mathcal{U}, w(u_i, U))\}$ (of the possible $3^{|\mathcal{U}|}$ or $4^{|\mathcal{U}|}$), storing the maximum number of sets that can be selected from $\{S_1, \ldots, S_j\}$ to form a partial solution $S' \subseteq \{S_1, \ldots, S_j\}$, so that for any element u_i it is $\max_{S_l \in S'}\{w(u_i, S_l) + w(u_i, U)\} \leq 1$. Letting the union operator $A \cup B$ transfer maximum inclusion, i.e. $w(u_i, A \cup B) = \max\{w(u_i, A), w(u_i, B)\}$, and substituting the check for $U \cap S_j = \emptyset$ by $\forall u_i \in U \cup S_j : w(u_i, U) + w(u_i, S_j) \leq 1$ in the above procedure, we can solve the Partial Set Packing instance in $O(mn4^n)$ time (and only $O(mn3^n)$ for even d).

Given a solution $S \subseteq \mathcal{S}$ to our Partial Set Packing instance, we will show that it corresponds to a solution for the original instance of d-Scattered Set. First observe that on any shortest path $v_0, \ldots, v_d$ between vertices v_0, v_d, we know that any vertex v_i will either be included in C, or both its neighbors v_{i-1}, v_{i+1} on the path will be included instead, as otherwise both edges adjacent to v_i are not covered by C.

Consider first the case where d is even. On the shortest path between two vertices v_0, v_d that are at distance d from each other there will be one (*middle*) vertex $v_{d/2}$ at distance $d/2$ from both and if $v_{d/2} \in C$ then the corresponding element will be partially included in both sets corresponding to v_0, v_d, while if $v_{d/2} \notin C$, each of the elements corresponding to its neighbors $v_{d/2-1}, v_{d/2+1}$ on the path will be completely included in one set each and thus both sets can be used in solution S. For two vertices v_0, v_{d-1} at distance $d-1$ from each other, there will be one vertex $v_{d/2}$ at distance $d/2$ from v_0 and $d/2-1$ from v_{d-1} and also one vertex $v_{d/2-1}$ at distance $d/2-1$ from v_0 and $d/2$ from v_{d-1}. If $v_{d/2} \in C$, then its corresponding element is included partially by $1/2$ in the set corresponding to vertex v_0 and completely by 1 in the set corresponding to vertex v_{d-1}. Otherwise, if $v_{d/2-1} \in C$, then its corresponding element is included completely by 1 in the set corresponding to vertex v_0 and partially by $1/2$ in the set corresponding to vertex v_{d-1}. Thus in both cases these two sets cannot be included together in S. The argument also holds if the distance between the two vertices is smaller than $d-1$.

Next, if d is odd, on the shortest path between two vertices v_0, v_d that are at distance d from each other there will be two middle vertices $v_{\lfloor d/2 \rfloor}, v_{\lceil d/2 \rceil}$ at distances $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil$ from each. Now, vertex $v_{\lfloor d/2 \rfloor}$ will be at distance $\lfloor d/2 \rfloor$ from v_0 and distance $\lceil d/2 \rceil$ from v_d and if $v_{\lfloor d/2 \rfloor} \in C$ its element will be included by $2/3$ in the set corresponding to v_0 and by $1/3$ in the set corresponding to v_d. Similarly, if $v_{\lceil d/2 \rceil} \in C$, its element will be included by $1/3$ in the set corresponding to v_0 and by $2/3$ in the set corresponding to v_d. Thus in both cases the two sets can be included together in S. For two vertices v_0, v_{d-1} at distance $d-1$ from each other, if vertex $v_{\lfloor d/2 \rfloor} \in C$, then its element will be included by $2/3$ in both sets corresponding to v_0, v_{d-1}, while if $v_{\lfloor d/2 \rfloor} \notin C$, then we have that both its neighbors $v_{\lfloor d/2 \rfloor - 1}$, $v_{\lfloor d/2 \rfloor + 1} \in C$. Now, the element corresponding to $v_{\lfloor d/2 \rfloor - 1}$ will be completely included by 1 in the set corresponding to v_0 and partially by $1/3$ in the set corresponding to v_{d-1}, while the element corresponding to $v_{\lfloor d/2 \rfloor + 1}$ will likewise be included partially by $1/3$ in the set corresponding to v_0 and completely by 1 in the set

corresponding to v_{d-1}. Thus in both cases, these two sets cannot be included together in S, while the argument also holds if the distance between the vertices is smaller than $d-1$.

As the number of sets in our Partial Set Packing instance is $m = |C| + |Y| \leq |V|$ and the number of elements is $n = |C| = \text{vc}$, the total running-time of our algorithm is bounded by $O^*(4^{\text{vc}})$ for odd d and $O^*(3^{\text{vc}})$ for even d. □

4.2.3 Tree-depth: Tight ETH-based lower bound

In this subsection we consider the unweighted version of the d-Scattered Set problem parameterized by td. We will first show the existence of an FPT algorithm of running-time $O^*(2^{O(\text{td}^2)})$ and then a tight ETH-based lower bound. Similarly to Subsection 3.2.3, we begin with a simple upper bound argument and the algorithm then follows from Theorem 51 and the relationship between d, td and tw:

Theorem 57. *Unweighted d-Scattered Set can be solved in time $O^*(2^{O(td^2)})$.*

Proof. As in the proof of Theorem 31, we can assume that $d \leq diam(G)$, where $diam(G)$ denotes the graph's diameter, because otherwise the problem is trivial. Hence, by Lemma 30 we have $d \leq 2^{\text{td}+1}$ and using Lemma 3 we can get $\text{tw} \leq \text{td}$. Then the algorithm of Theorem 51 which runs in time $O^*(d^{\text{tw}})$ gives the desired running-time. □

Next we show a lower bound matching Theorem 57, based on the ETH, using a reduction from 3-SAT and a construction similar to the one used in Subsection 4.2.1.

Construction: Given an instance ϕ of 3-SAT on n variables and m clauses, where we can assume that $m = O(n)$ (by the Sparsification Lemma, see [61]), we will create an instance $[G = (V, E)]$ of the unweighted d-Scattered Set problem where $d = 6 \cdot c^{\sqrt{n}}$ for an appropriate constant c (to simplify notation, we assume without loss of generality that $\sqrt{n}$ is an integer). We first group the clauses of ϕ into $\sqrt{n}$ equal-sized groups $F_1, \ldots, F_{\sqrt{n}}$ and as a result, each group involves $O(\sqrt{n})$ variables, with $2^{O(\sqrt{n})}$ possible assignments to the variables of each group. We select c appropriately so that each group F_i has at most $c^{\sqrt{n}}$ possible partial assignments ϕ_j^i for the variables of clauses in F_i.

We then create for each $i \in \{1, \ldots, \sqrt{n}\}$, a set P_i of at most $c^{\sqrt{n}}$ vertices $p_1^i, \ldots, p_{c^{\sqrt{n}}}$, such that each vertex of P_i represents a partial assignment to the variables of F_i that satisfies all clauses of F_i. We then create for each $i \in \{1, \ldots, \sqrt{n}\}$ a pair of vertices a_i, b_i and we connect a_i to each vertex p_l^i by a path of length $c^{\sqrt{n}} + l$, while b_i is connected to each vertex p_l^i by a path of length $2 \cdot c^{\sqrt{n}} - l$. Now each P_i contains all a_i, b_i and $p_l^i, i \in \{1, \ldots, c^{\sqrt{n}}\}$.

Finally, for every two *non-conflicting* partial assignments ϕ_l^i, ϕ_o^j, with $l, o \in [1, c^{\sqrt{n}}]$ and $i, j \in [1, \sqrt{n}]$, i.e. two partial assignments that do not assign conflicting values to any variable, we create a vertex $u_{l,o}^{i,j}$ that we connect to vertices a_i, b_i and a_j, b_j: if $p_l^i \in P_i$ is the vertex corresponding to ϕ_l^i and $p_o^j \in P_j$ is the vertex corresponding to ϕ_o^j, then vertex $u_{l,o}^{i,j}$ is connected to a_i by a path of length $5 \cdot c^{\sqrt{n}} - l$ and to b_i by a path of length $4 \cdot c^{\sqrt{n}} + l$, as well as to a_j by a path of length $5 \cdot c^{\sqrt{n}} - o$ and to b_j by a path of length $4 \cdot c^{\sqrt{n}} + o$. Next, for every pair i, j we make two vertices $g_{i,j}, g'_{i,j}$. We connect $g_{i,j}$ to all vertices $u_{l,o}^{i,j}$ (for any l, o) by a single edge and also $g_{i,j}$ to $g'_{i,j}$ by a path of length $6 \cdot c^{\sqrt{n}} - 1$. This concludes our construction and Figure 4.5 provides an illustration.

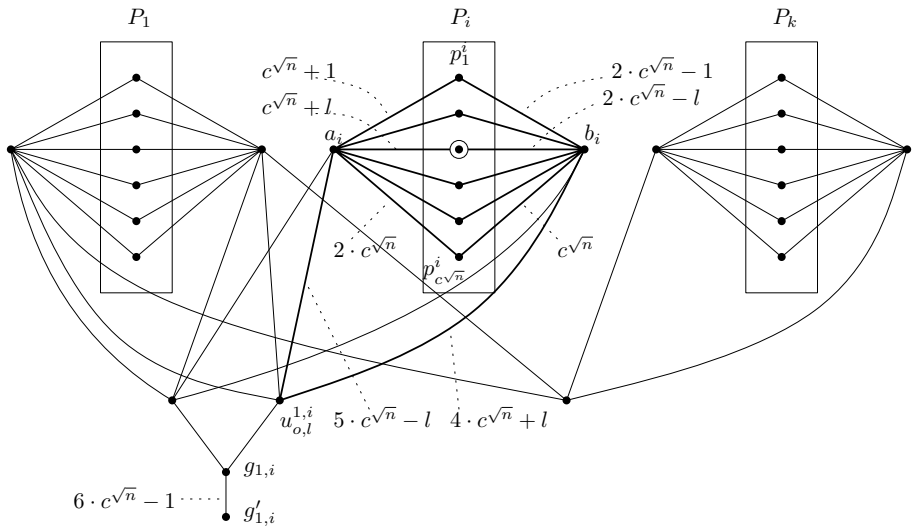


Figure 4.5: A general picture of graph G, where straight lines imply paths of length equal to the number indicated by dotted lines, while the circled vertex is p^i_l.

Lemma 58. *If ϕ has a satisfying assignment, then there exists a $6\cdot c^{\sqrt{n}}$-scattered set in G of size $\sqrt{n}+2\binom{\sqrt{n}}{2}=n$.*

Proof. Consider the satisfying assignment for ϕ and let $\phi^i_{l_i}$, with $l_i \in [1, c^{\sqrt{n}}]$ and $i \in [1,\sqrt{n}]$, be the restriction of that assignment for all variables appearing in clauses of group F_i. We claim the set K, consisting of all vertices $p^i_{l_i}$ corresponding to $\phi^i_{l_i}$, all vertices $g'_{i,j}$ and all $u^{i,j}_{l,o}$ vertices for which we have selected p^i_l and p^j_o (all these vertices exist, as the corresponding partial assignments are non-conflicting), is a d-scattered set for G of size $|K|=\sqrt{n}+2\binom{\sqrt{n}}{2}=n$: all selected vertices $p^i_{l_i}$ are at distance $c^{\sqrt{n}}+l_i+5\cdot c^{\sqrt{n}}-l_i=6\cdot c^{\sqrt{n}}$ via a_i and distance $2\cdot c^{\sqrt{n}}-l_i+4\cdot c^{\sqrt{n}}+l_i=6\cdot c^{\sqrt{n}}$ via b_i from any selected vertex $u^{i,j}_{l_i,l_j}$, while every selected $u^{i,j}_{l_i,l_j}$ is at distance $6\cdot c^{\sqrt{n}}-1+1=6\cdot c^{\sqrt{n}}$ from every selected $g'_{i,j}$. □

Lemma 59. *If there exists a $6\cdot c^{\sqrt{n}}$-scattered set in G of size $\sqrt{n}+2\binom{\sqrt{n}}{2}=n$, then ϕ has a satisfying assignment.*

Proof. Let $S\subseteq V$ be the $6\cdot c^{\sqrt{n}}$-scattered set in G, with $|S|=n$. For every pair $i,j\in[1,\sqrt{n}]$, set S cannot contain more than one vertex from the paths between $g_{i,j}$ and a_i,b_i,a_j,b_j, as the distance between any pair of such vertices is always $<2\cdot 6\cdot c^{\sqrt{n}}$ (due to the single edges between $g_{i,j}$ and any $u^{i,j}_{l_i,l_j}$). Likewise, set S cannot contain more than two vertices from the paths between $g'_{i,j}$ and a_i,b_i,a_j,b_j, as the maximum sum of distances between any three such vertices is $<3\cdot 6\cdot c^{\sqrt{n}}$. Since $|S|=\sqrt{n}+2\binom{\sqrt{n}}{2}$, set S must also contain $\sqrt{n}$ other vertices and due to the distance between any pair of vertices p^i_l,p^i_o from the same group P_i being $<4\cdot\sqrt{n}$, there must be one selection from each group P_i. Furthermore, for two such selections $p^i_{l_i},p^j_{l_j}$, the only option for the other two selections (for this pair of groups i,j) is to select vertices $g'_{i,j}$ and $u^{i,j}_{l_i,l_j}$, since the distances from $p^i_{l_i},p^j_{l_j}$ to $u^{i,j}_{l_i,l_j}$ (through a_i,b_i,a_j,b_j) will only be equal to $6\cdot c^{\sqrt{n}}$ if these selections (and indices) match, with the only remaining option at distance $6\cdot c^{\sqrt{n}}$ (for any choice of $u^{i,j}_{l_i,l_j}$) being vertex $g'_{i,j}$. □

Lemma 60. *The tree-depth of G is $2\sqrt{n} + \lceil \log(6 \cdot c^{\sqrt{n}}) \rceil + 1 = O(\sqrt{n})$.*

Proof. We again employ the alternative definition of tree-depth used in Lemma 30. Consider graph G after removal of all vertices $a_i, b_i, \forall i \in [1, \sqrt{n}]$. The graph now consists of $\sqrt{n} \cdot c^{\sqrt{n}}$ paths of length $< 3 \cdot c^{\sqrt{n}}$ through each vertex in P_i and $\binom{\sqrt{n}}{2}$ trees, considered rooted at each vertex $g_{i,j}$. The maximum distance in each such tree between a leaf and its root is $6 \cdot c^{\sqrt{n}} - 1$ (for vertex $g'_{i,j}$) and the claim then follows, as paths of length n have tree-depth exactly $\lceil \log(n+1) \rceil$ (this can be shown by repeatedly removing the middle vertex of each path). By the definition of tree-depth, after removal of $2\sqrt{n}$ vertices from G, the maximum tree-depth of each resulting disconnected component is $\lceil \log(6 \cdot c^{\sqrt{n}}) \rceil = \lceil \sqrt{n} \cdot \log(c) + \log(6) \rceil$. □

Theorem 61. *If unweighted d-Scattered Set can be solved in $2^{o(td^2)} \cdot n^{O(1)}$ time, then* 3-SAT *can be solved in $2^{o(n)}$ time.*

Proof. Suppose there is an algorithm for d-Scattered Set with running-time $2^{o(\mathrm{td}^2)}$. Given an instance ϕ of 3-SAT, we use the above construction to create an instance $[G = (V, E)]$ of d-Scattered Setwith $d = 6 \cdot c^{\sqrt{n}}$, in time $O(\sqrt{n} \cdot c^{\sqrt{n}} + c^{2\sqrt{n}})$. As, by Lemma 60, we have $\mathrm{td}(G) \leq O(\sqrt{n})$, using the supposed algorithm for d-Scattered Set we can decide whether ϕ has a satisfying assignment in time $2^{o(\mathrm{td}^2)} \cdot n^{O(1)} = 2^{o(n)}$. □

4.3 Treewidth revisited: FPT approximation scheme

In this section we present an FPT-AS for d-Scattered Set parameterized by tw. Given as input an edge-weighted graph $G = (V, E)$, $k \in \mathbb{N}^+$, $d \geq 2$ and an arbitrarily small error parameter $\epsilon > 0$, our algorithm is able to return a set K, such that any pair $v, u \in K$ are at distance $d(v, u) \geq \frac{d}{1+\epsilon}$, in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, if a d-scattered set of size $|K| = k$ exists in G. As was the case in Section 3.3, our algorithm makes use of the technique of [73] for approximating problems that are W-hard by treewidth.

The rounding technique as applied in [73] employs randomization and an extensive analysis to procure the bounds on the propagation of error, while we only require a deterministic adaptation of the rounding process without making use of the advanced machinery there introduced, as for our particular case, the bound on the rounding error can be straightforwardly obtained. The main tool we require is the following definition of an addition-rounding operation, denoted by $\oplus$:[3] for two non-negative numbers x_1, x_2, we define $x_1 \oplus x_2 \coloneqq 0$, if $x_1 = x_2 = 0$. Otherwise, we set $x_1 \oplus x_2 \coloneqq (1+\delta)^{\lfloor \log_{(1+\delta)}(x_1 + x_2) \rfloor}$.

For our purposes, the integers we would like to approximately store are the states $s_j \in [1, d-1]$, representing the distance of a vertex v_j in some bag X_i of the tree decomposition to the closest selection in the d-scattered set K, during computation of the dynamic programming algorithm. Instead of allowing use of all $d-1$ exact values to signify the states, we will instead adopt rounded approximations, that are inductively computed and propagated, while keeping accumulation of error bounded as well. Let $\Sigma_\delta \coloneqq \{0\} \cup \{(1+\delta)^l | l \in \mathbb{N}\}$. Intuitively, Σ_δ is the set of rounded states that our modified algorithm may use. Of course, Σ_δ as defined is infinite, but we will only consider the set of values that are at most d, denoted by Σ_δ^d. In this way, the size of Σ_δ^d is reduced to $\log_{(1+\delta)}(d)$, for an

[3]The [73] version uses a randomly selected real number < 1 in the sum within the logarithm, while our application can be seen as setting this value to 0, avoiding randomization altogether, as mentioned in Section 3.3. We also omit references to Addition Trees in this section and only make use of the $\oplus$ operator.

appropriate choice of δ, to be specified later. For all nodes i of the tree decomposition, we now have $D_i[\sigma_0, \ldots, \sigma_t] : (\Sigma_\delta^d)^{t+1} \mapsto \mathbb{N}^0$, with $0 \leq t < \text{tw}$.

Modifications: Our approximation algorithm will be a modification of the exact dynamic programming for d-Scattered Set, given in Subsection 4.1.2. For the approximation algorithm, we will make use of an adaptation of this algorithm of Theorem 51, that works for the maximization version of the problem instead of the counting version (albeit not optimally). We first describe the necessary modifications to the counting version and then the subsequent changes for use of our rounded values. In fact, any "simple" version of the exact algorithm that does not involve counting the number of solutions could be used as the basis for our modified approximation algorithm.

The algorithm for the maximization version needs the following changes: for a leaf node i we set $D_i[s_0] \coloneqq 1$, if $s_0 = 0$, and 0 otherwise. For an introduce node i, we also add a $+1$ to the values of previously computed entries if $s_{t+1} = 0$ and the same conditions hold as in the counting version, while a value of 0 for invalid state-representations is substituted by an arbitrarily large negative value $-\infty$. For forget nodes i we now compare all previous partial solutions to retain the maximum over all states of the forgotten vertex, instead of computing their sum, while for join nodes, we also substitute taking the sum by taking the maximum, with multiplication also substituted by addition of entries from the previous tables (i.e. we move our computations from the sum-product ring to the max-sum semiring), as well as subtracting from each such computation the number of vertices of zero state for the given entry (that would be counted twice).

We next explain the necessary modifications to the exact algorithm for use of the rounded states $\sigma \in \Sigma_\delta^d$. Consider a node i introducing vertex v_{t+1}: for a new entry to describe a proper extension to some previously computed partial solution, if the new vertex is of state $s_{t+1} \in [1, d-1]$ in the new entry, then there must be some vertex $v_j \in X_i$, such that $s_{t+1} \leq d(v_{t+1}, v_j) + s_j$ (the one for which this sum is minimized), i.e. we require that the new state of the introduced vertex matches its distance to some other vertex in the bag plus the state of that vertex (being the one responsible for connecting v_{t+1} to the partial solution). The rounded state σ_{t+1} for v_{t+1} must now satisfy: $\sigma_{t+1} \leq d(v_{t+1}, v_j) \oplus \sigma_j$, where $\oplus$ is the operator defined above.

Further, states are now considered low if $0 < \sigma \leq \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}$, while, from a set of already computed states σ', the symmetrical (around $d/2$) state $\bar{\sigma}$ for a given low state σ is defined as the minimum state σ' for which $\sigma + \sigma' \geq \frac{d}{(1+\epsilon)}$. Thus, for a node introducing vertex v_{t+1} with state $\sigma_{t+1} = 0$, we require that $\forall v_j \in X_{i-1}$ with $\sigma_j = 0$, it is $d(v_{t+1}, v_j) \geq \frac{d}{(1+\epsilon)}$, and $\forall v_j \in X_{i-1}$ with $d(v_{t+1}, v_j) \leq \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}$, it is $\sigma_j \leq d(v_{t+1}, v_j)$ and $\sigma' = \bar{\sigma}$ for $D_i[\sigma_0, \ldots, \sigma_{t+1}] \coloneqq D_{i-1}[\sigma'_0, \ldots, \sigma'_t] + 1$. Finally, for join nodes, we arbitrarily choose the computed states for the table of one of the children nodes to represent the new entries and again use $\bar{\sigma}$ to identify the symmetrical of each low state (from the other node's table).

Moreover, we require that the tree decompositions on which our algorithm is to be applied are rooted and of maximum height $O(\log n)$ and for this we again use Theorem 39. The following lemma employs the transformation of [13] to bound the error of any value calculated in this way, based on an appropriate choice of δ and therefore set Σ_δ^d of available values, by relating the rounded states σ computed at any node to the states s that the exact algorithm would use at the same node instead.

Lemma 62. *Given ϵ and a tree decomposition $(\mathcal{X}, T)$ with $T = (I, F), \mathcal{X} = \{X_i | i \in I\}$, where T is rooted, binary and of height $O(\log n)$, there exists a constant C, such that for all rounded states $\sigma_j \in \Sigma_\delta^d$ it is $\sigma_j \geq \frac{s_j}{(1+\epsilon)}, \forall v_j \in X_i, \forall i \in I$, where $\delta = \frac{\epsilon}{C \log n}$.*

Proof. First, observe that for any rounded state σ calculated using the $\oplus$ operator we have $\sigma \leq s$, where s is the state the exact algorithm would use instead. Let h be the maximum depth of the recursive computations of any state σ we may require. We now want to show by induction on h that it is always $\log_{1+\delta}(\frac{s}{\sigma}) \leq h$. For $h = 1$ and only one addition $\sigma = 0 \oplus d_1$, for some distance d_1 with $s = 0 + d_1$, we want $\log_{(1+\delta)}(\frac{s}{\sigma}) \leq 1$. It is indeed $\log_{(1+\delta)}(\frac{s}{\sigma}) = \log_{(1+\delta)}(d_1) - \lfloor \log_{(1+\delta)}(d_1) \rfloor \leq 1$.

For the inductive step, let $\sigma_3 = \sigma_2 \oplus d_2$ and $s_3 = s_2 + d_2$ be the final rounded and exact values (at height h), for some distance d_2 and previous values σ_2, s_2 (for $h - 1$). It is $\log_{(1+\delta)}(\frac{s_3}{\sigma_3}) = \log_{(1+\delta)}(\frac{s_2+d_2}{(1+\delta)^{\lfloor \log_{(1+\delta)}(\sigma_2+d_2) \rfloor}}) = \log_{(1+\delta)}(s_2) + \log_{(1+\delta)}(1 + \frac{d_2}{s_2}) - \lfloor \log_{(1+\delta)}(\sigma_2) + \log_{(1+\delta)}(1 + \frac{d_2}{\sigma_2}) \rfloor$. This, after removal of the floor function, is $\leq \log_{(1+\delta)}(s_2) + \log_{(1+\delta)}(1 + \frac{d_2}{s_2}) - (\log_{(1+\delta)}(\sigma_2) + \log_{(1+\delta)}(1 + \frac{d_2}{\sigma_2})) + 1 = \log_{(1+\delta)}(s_2) - \log_{(1+\delta)}(\sigma_2) + \log_{(1+\delta)}(1 + \frac{d_2}{s_2}) - \log_{(1+\delta)}(1 + \frac{d_2}{\sigma_2}) + 1$. The claim then follows, because $\log_{(1+\delta)}(s_2) - \log_{(1+\delta)}(\sigma_2) = \log_{(1+\delta)}(\frac{s_2}{\sigma_2}) \leq h - 1$ by the inductive hypothesis, while also $\log_{(1+\delta)}(1 + \frac{d_2}{s_2}) - \log_{(1+\delta)}(1 + \frac{d_2}{\sigma_2}) \leq 0$, as $\sigma_2 \leq s_2$.

Thus we have $\log_{(1+\delta)}(\frac{s}{\sigma}) \leq h$, from which we get $\frac{s}{\sigma} \leq (1 + \delta)^h$. For $\sigma \geq \frac{s}{(1+\epsilon)}$, we require that $(1 + \delta)^h \leq (1 + \epsilon)$, or $h \leq \log_{(1+\delta)}(1 + \epsilon) = \frac{\log_2(1+\epsilon)}{\log_2(1+\delta)}$, that gives $h \leq \frac{\epsilon}{\delta}$, for $\epsilon, \delta \approx 0$, or $\delta \leq \frac{\epsilon}{h}$. Next, observe that during the computations of the algorithm, the maximum depth h of any computation can only increase by one if some vertex is introduced in the tree decomposition, as paths to and from it become available. This means no inductive computation we require can be of depth larger than the height of the tree decomposition T, giving $h = C \log n$ for some constant C. □

Theorem 63. *There is an algorithm which, given an edge-weighted instance of d-*Scattered Set *$[G, k, d]$, a tree decomposition of G of width tw and a parameter $\epsilon > 0$, runs in time $O^*((tw/\epsilon)^{O(tw)})$ and finds a $d/(1+\epsilon)$-scattered set of size k, if a d-scattered set of the same size exists in G.*

Proof. Naturally, our modified algorithm making use of these rounded values to represent the states will not perform the same computations as the exact version given in Subsection 4.1.2. The new statement of correctness, taking into account the approximate values now computed (and the switch to the maximization version), is the following:

$$\forall i \in I, \forall (\sigma_0, \ldots, \sigma_t) \in (\Sigma_\delta^d)^{t+1}, 0 \leq t < \text{tw}: \tag{4.13}$$

$$\{D_i[\sigma_0, \ldots, \sigma_t] = |K| : K \subseteq V_i \setminus X_i \cup \{v_l \in X_i | \sigma_l = 0\}: \tag{4.14}$$

$$\{\forall u, w \in K : d(u, w) \geq \frac{d}{(1+\epsilon)}\} \wedge \tag{4.15}$$

$$\wedge \{\forall v_j \in X_i | 0 < \sigma_j \leq \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}, \forall u, w \in K | d(u, v_j) \leq d(w, v_j): \tag{4.16}$$

$$d(u, v_j) \geq \sigma_j \wedge d(w, v_j) \geq \frac{d}{(1+\epsilon)} - \sigma_j\} \wedge \tag{4.17}$$

$$\wedge \{\forall v_j \in X_i | \sigma_j > \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}, \forall u \in K : d(u, v_j) \geq \sigma_j\} \vee \tag{4.18}$$

$$\vee D_i[\sigma_0, \ldots, \sigma_t] = -\infty\}. \tag{4.19}$$

In words, the above states that for every node i and all possible state-configurations $(\sigma_0, \ldots, \sigma_t) \in (\Sigma_\delta^d)^{t+1}$ (4.13), table entry $D_i[\sigma_0, \ldots, \sigma_t]$ contains the size of a subset K of V_i (that includes vertices $v_l \in X_i$ of state $s_l = 0$) (4.14), such that the distance between every pair of vertices u, w in K is at least $d/(1+\epsilon)$ (4.15), for every vertex $v_j \in X_i$ of low state $\sigma_j \leq \lfloor d/2 \rfloor/(1+\epsilon)$ and a pair of vertices u, w from K with u closer to v_j than w (4.16), its distance to u is at least equal to its state σ_j and its distance to w is at least $d/(1+\epsilon) - \sigma_j$ (4.17), while for every vertex $v_j \in X_i$ of high state $\sigma_j > \lfloor d/2 \rfloor/(1+\epsilon)$ and a vertex u from K, its state σ_j is at most its distance from any vertex u (4.18), or if there is no such K, we have $D_i[\sigma_0, \ldots, \sigma_t] = -\infty$ for this entry (4.19). This is shown by induction on the nodes $i \in I$:

- Leaf node i with $X_i = \{v_0\}$: This is the base case of our induction and the initializing values of 1 for $\sigma_0 = 0$ and 0 for $\sigma_0 > 0$ are indeed the correct sizes for K.

- Introduce node i with $X_i = X_{i-1} \cup \{v_{t+1}\}$: For entries with $0 < \sigma_j \leq \lfloor d/2 \rfloor/(1+\epsilon)$, validity of (4.15,4.18) is not affected, while for (4.16-4.17): it is $\sigma_{t+1} = \sigma_j \oplus d(v_{t+1}, v_j)$ for some vertex $v_j \in X_{i-1}$, for which, by the induction hypothesis we have that $\sigma_j \leq d(u, v_j)$ and $d(w, v_j) \geq d/(1+\epsilon) - \sigma_j$, where u is the closest selection to v_j and w the second closest. To see the same holds for v_{t+1}, observe that $\sigma_{t+1} \leq d(v_{t+1}, v_j) + d(u, v_j) = d(u, v_{t+1})$ and $d(w, v_j) \geq d/(1+\epsilon) - \sigma_j \Rightarrow \sigma_j + d(v_{t+1}, v_j) \geq d/(1+\epsilon) - d(w, v_j) + d(v_{t+1}, v_j) \Rightarrow \sigma_j + d(v_{t+1}, v_j) \geq d/(1+\epsilon) - d(w, v_{t+1}) \Rightarrow \sigma_{t+1} \geq d/(1+\epsilon) - d(w, v_{t+1})$.

  For entries with $\sigma_{t+1} > \lfloor d/2 \rfloor/(1+\epsilon)$, validity of (4.15-4.17) is not affected, while for (4.18): it is $\sigma_{t+1} \leq d(v_{t+1}, v_j) + \sigma_j$ for some $v_j \in X_{i-1}$, for which we have $\sigma_j \leq d(u, v_j)$ and thus also $\sigma_{t+1} \leq d(v_{t+1}, v_j) + d(v_j, u) = d(v_{t+1}, u)$.

  For entries with $\sigma_{t+1} = 0$, observe that the low states σ_j of vertices $v_j \in X_{i-1}$ in the new entry with $d(v_{t+1}, v_j) \leq \lfloor d/2 \rfloor/(1+\epsilon)$ would need to be $\sigma_j \leq d(v_{t+1}, v_j)$ and also correspond to the minimum high original state σ'_j such that $\sigma_j + \sigma'_j \geq d/(1+\epsilon)$, for which partial solution it is $d(v_j, u) \geq \sigma'_j, \forall u \in K$ and thus $d(v_j, u) \geq d/(1+\epsilon) - \sigma_j$ (4.16-4.17). For high states σ_j of vertices $v_j \in X_{i-1}$, it is $d(v_{t+1}, v_j) \geq \sigma_j$ (4.18) and finally, for (4.15), if there was some $u \in K$ such that $u \notin X_{i-}$ and $d(u, v_{t+1} < d/(1+\epsilon)$, then there must be some $v_j \in X_{i-1}$ of new state σ_j and previous state σ'_j (on the path between u and v_{t+1}) for which $\sigma_j + \sigma'_j < d/(1+\epsilon)$, contradicting the

requirement for introduction of v_{t+1} with $\sigma_{t+1} = 0$: it is $d(u, v_{t+1}) < d/(1+\epsilon) \Rightarrow d(u, v_j) + d(v_{t+1}, v_j) < d/(1+\epsilon) \Rightarrow \sigma_j + \sigma'_j < d/(1+\epsilon)$, as it must be $\sigma'_j \leq d(u, v_j)$ and also $\sigma_j \leq d(v_{t+1}, v_j)$.

- Forget node i with $X_i = X_{i-1} \setminus \{v_{t+1}\}$: No modification to the exact dynamic programming affects the correctness of (4.13-4.18), as, the right number is indeed the maximum over all states for v_{t+1}.

- Join node i with $X_i = X_{i-1} = X_{i-2}$: For (4.15), if there was a pair $u \in K \cap V_{i-1} \setminus X_{i-1}$, $w \in K \cap V_{i-2} \setminus X_{i-2}$ with $d(u, w) < d/(1+\epsilon)$, then there must be some vertex $v_j \in X_i$ (on the path between the two) for which $\sigma_j + \bar{\sigma_j} < d/(1+\epsilon)$ (as above). For (4.16-4.18), observe that for vertices of low state σ_j, lines (4.16-4.17) must have been true for either $i-1$ or $i-2$ and (4.18) for the other, while for vertices v_j of high state σ_j, it again suffices that (4.18) must have been true for both.

For a node $i \in I$, let $U_i(k, s_0, \ldots, s_t) = \{K \subseteq V_i | K \cap X_i = \{v_j \in X_i | s_j = 0\}\}$ be the set of all d-scattered sets in G_i of size k for this state-configuration $(s_0, \ldots, s_t)$ (as in the proof of Theorem 51), $U_i(k, \sigma_0, \ldots, \sigma_t) = \{K \subseteq V_i | K \cap X_i = \{v_j \in X_i | \sigma_j = 0\}$ be the set of all subsets K of V_i of size k for the rounded state-configuration $(\sigma_0, \ldots, \sigma_t)$ (computed by our approximation algorithm) and $U_i(k, \frac{s}{1+\epsilon})$ be the set of all $d/(1+\epsilon)$-scattered sets of size k in G_i. Consider a set $K \in U_i(k, s_0, \ldots, s_t)$ and let $(\sigma_0, \ldots, \sigma_t)$ be the state-configuration resulting from rounding each s_j down to its closest integer power of $(1+\delta)$, or $\sigma_j = (1+\delta)^{\lfloor \log_{(1+\delta)}(s_j) \rfloor}, \forall j \in [0, t]$. As $|K| = k$ and for any pair $u, w \in K$, we have $d(u, w) \geq d > \frac{d}{(1+\epsilon)}$, we want to show that the requirements of $(\sigma_0, \ldots, \sigma_t)$ also hold for K. By Lemma 62, we know that $s_j \geq \sigma_j \geq \frac{s_j}{(1+\epsilon)}$ for all $j \in [0, t]$. Now, for each $\sigma_j \in (\sigma_0, \ldots, \sigma_t)$, $\sigma_j \leq s_j$ gives $\sigma_j \leq d(u, v_j)$ for the closest $u \in K$ to v_j, while if also $\sigma_j \leq \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}$, then $\frac{s_j}{(1+\epsilon)} \leq \frac{\lfloor d/2 \rfloor}{(1+\epsilon)}$ and $s_j \leq \lfloor d/2 \rfloor$ (i.e. s_j is also low) and we have that $d - s_j \leq d(w, v_j)$ for $w \in K$ being the second closest to v_j, from which we get $\frac{d}{(1+\epsilon)} - \frac{s_j}{(1+\epsilon)} \leq \frac{d(w,v_j)}{(1+\epsilon)} \Rightarrow \frac{d}{(1+\epsilon)} - \sigma_j \leq \frac{d(w,v_j)}{(1+\epsilon)} \leq d(w, v_j)$, i.e. state-configuration $(\sigma_0, \ldots, \sigma_t)$ also holds for set K. This means $K \in U_i(k, \sigma_0, \ldots, \sigma_t)$ and thus we have $U_i(k, s_0, \ldots, s_t) \subseteq U_i(k, \sigma_0, \ldots, \sigma_t)$. Further, since for any K our approximation algorithm will compute, it is $d(u, w) \geq d/(1+\epsilon), \forall u, w \in K$, we also have $U_i(\sigma_0, \ldots, \sigma_t) \subseteq U_i(k, \frac{s}{(1+\epsilon)})$. Due to these considerations, if a d-scattered set of size k exists in G, our algorithm will be able to return a set K with $|K| = k$, that will be a $d/(1+\epsilon)$-scattered set of G.

The algorithm is then the following: first, according to the statement of Lemma 62, we select $\delta = \frac{\epsilon}{C \log n}$, that we use to define set Σ^d_δ and then we use the algorithm of Theorem 51, modified as described above, on the bounded-height transformation of nice tree decomposition $(\mathcal{X}, T)$. Correctness of the algorithm and justification of the approximation bound are given above, while the running-time crucially depends on the size of Σ^d_δ being $|\Sigma^d_\delta| = O(\log_{(1+\delta)} d) = O(\frac{\log d}{\log(1+\delta)}) = O(\frac{\log d}{\delta})$, where we used the approximation $\log(1+\delta) \approx \delta$ for sufficiently small δ (i.e. sufficiently large n). This gives $O(\log n/\epsilon)^{O(\mathrm{tw})}$ and the statement is then implied by Lemma 2.

As a final note, observe that due to the use of the $\lfloor \cdot \rfloor$ function in the definition of our $\oplus$ operator, all our values will be rounded *down*, in contrast to the original version of the technique (from [73]), where depending on a randomly chosen number ρ, the values could be rounded either down or up. This means there will be some value x, such that $x \oplus 1 = x$, or $(1+\delta)^x = (1+\delta)^{\lfloor \log_{(1+\delta)}(x+1) \rfloor}$ (we would have $x \approx 1/\delta$). One may be tempted

to conceive of a pathological instance consisting of a long path on n vertices and $d >> x$, along with a simple path decomposition for it (that is essentially of the same structure), where the computations for each rounded state σ would "get stuck" at this value x. In fact, the transformation of [13] would give a tree decomposition of height $O(\log n)$ for this instance, whose structure would be the following: the leaf nodes would correspond to one vertex of the path each, while at (roughly) each height level i, sub-paths of length 2^i would be joined together. Thus each join node t that corresponds to some sub-path of length 2^i (let $X_t = \{a, b, c, d\}$) would have two child branches, consisting of two forget nodes, two introduce nodes and a previous join node on each side (let these be $t-1, t-2$), computing sub-paths of length 2^{i-1} (with $X_{t-1} = \{a, a', b', b\}$ and $X_{t-2} = \{c, c', d', d\}$). The vertices forgotten at each branch would be the middle vertices of the sub-path of length 2^{i-1} already computed at the previous join node of this branch (i.e. a', b' for the $t-1$ side and c', d' for the other), while the introduced vertices would be the endpoints of the sub-path of length 2^{i-1} computed at the other branch attached to this join node (i.e. c, d for the $t-1$ side and a, b for the other). In this way, in each branch (and partial solution) there will be one vertex (c or b) for which the rounded state would need to be $\leq 1\oplus$ the rounded state σ of some neighbor (b or c) and one vertex (d or a) for which the $\oplus$ operator would be applied between the state σ of some non-adjacent vertex (a, b or c for the $t-1$ side and c, d or b for the other) and their distance (e.g. $d(b, d)$ and $d(c, a)$), these being at least 2^{i-1}. In this way, the algorithm will not have to compute any series of rounded states sequentially by $\oplus 1$ and as, by Lemma 62, we have that for all nodes $i \in I$ and vertices $v_j \in X_i$, it is $\sigma_j \geq \frac{s_j}{(1+\epsilon)}$, for all $\sigma_j \in \Sigma_\delta^d$, the rounded states used by the algorithm for these introduce/join nodes will never be more than a factor of $(1+\epsilon)$ from the ones used by the exact algorithm on the same tree decomposition. $\square$

5

On the (Super-)Polynomial (In-)Approximability of d-Scattered Set

In this chapter we revisit the d-Scattered Set problem, yet our focus here lies on the aspects of the problem that relate more to its approximability than the sensitivity of its complexity to restrictions on the structure of the input. We consider running-times expressed as functions of n, the size of the instance, along with other relevant parameters, while we are interested in both polynomial, as well as super-polynomial (i.e. exponential) running-times. Furthermore, we concentrate on the unweighted variant of the problem, where the distance is measured only by the number of edges.

Following inspection of the problem's complexity with respect to the most commonly studied structural parameters, we are now interested in identifying the optimal relationship between the running-time of approximation algorithms, expressed as a function of the size of the instance n, the distance parameter d, and the best achievable approximation ratio ρ for the problem, in order to involve the size of the input and the quality of the solutions into play as parameters in our investigations.

Subsequent to this our perspective shifts in terms of the types of results we explore: in the previous chapters where our viewpoint was parameterized, our aim was to identify the correct values of running-times for each parameter under consideration, showing that these are in a certain sense optimal by providing matching upper and lower bounds, or concurring algorithmic and hardness results. Dealing with approximation our aim remains with securing tight bounds that stem from algorithmic and hardness results that concur, yet it is now the functions describing the approximation ratios that we are interested in precisely characterizing, through identification of upper/lower bounds on ratios achievable in strictly polynomial running-times. These are still influenced by restrictions to the structure of the input, as for our results we consider graphs of bounded degree and bipartite graphs, that are of special interest in terms of independence.

Before moving on to describe our results, we note that these may be dependent on the parity of our distance parameter d as being even or odd. Both our running-times and ratios can be affected by this peculiarity of the problem that, intuitively, arises due to the (non)existence of a *middle* vertex on a path of length d between two endpoints. If d is even then such a vertex can exist at equal distance $d/2$ from any number of vertices in the solution, while if d is odd there can be no vertex at equal distance from any pair of vertices in the solution. This idiosyncrasy can change the way in which both our algorithms and

hardness constructions work (already seen in the algorithm of Theorem 56) and in some cases even entirely alters the complexity of the problem (e.g. in the results of [42]).

Our results: Section 5.1 concerns super-polynomial running-times, presenting first an exact exponential-time algorithm for d-Scattered Set of complexity $O^*((ed)^{\frac{2n}{d}})$ and then considering the inapproximability of the problem within the same range. We show that no ρ-approximation algorithm can take time (roughly) $2^{\frac{n^{1-\epsilon}}{\rho d}}$ for even d and $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ for odd d, under the (randomized) ETH. This is complemented by (almost) matching ρ-approximation algorithms of running-times $O^*((e\rho d)^{\frac{2n}{\rho d}})$ for even d and $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ for odd d. We note that the current state-of-the-art PCPs[1] are unable to distinguish between optimal running-times of the form $2^{n/\rho}$ and $\rho^{n/\rho}$ for ρ-approximation algorithms, due to the poly-logarithmic factor added by even the most efficient constructions and we thus do not focus on the poly-logarithmic factors differentiating our upper and lower bounds. These results provide a complete characterization of the optimal relationship between the worst-case approximation ratio ρ achievable for d-Scattered Set by any algorithm, its running-time and the distance parameter d, *for any point in the trade-off curve*, in a similar manner as was done for Independent Set in [26, 36] (see also [21, 23]), by also considering the range of possible values for d. We observe that the distance parameter d acts as a scaling factor for the size of the instance, whereby the problem becomes easier when vertices are required to be much further apart, a feature counterbalanced by the chosen approximation ratio ρ, with small values guaranteeing the quality of the produced solutions, yet also negatively impacting on the exponent of the running-time.

Section 5.2 follows this up by considering strictly polynomial running-times. We first show that there is no polynomial-time approximation algorithm for d-Scattered Set with ratio $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ in graphs of maximum degree Δ. This is the first lower bound that considers Δ and generalizes the known $\Delta^{1-\epsilon}$-inapproximability of Independent Set (see Theorem 5.2 of [26], restated here as Theorem 10, as well as [3]). Maximum vertex degree Δ plays an important role in the context of independence (e.g. [8, 38, 57]) and was specifically studied for d-Scattered Set in [43], where polynomial-time $O(\Delta^{d-1})$- and $O(\Delta^{d-2}/d)$-approximations are given. We improve upon this by showing that any greedy approximation algorithm in fact achieves a ratio of $O(\Delta^{\lfloor d/2 \rfloor})$, also matching our lower bound. Finally, we turn to bipartite graphs and show that d-Scattered Set can be approximated within a factor of $2\sqrt{n}$ in polynomial time also for even values of d, matching its known $n^{1/2-\epsilon}$-inapproximability from [42] and complementing the known $\sqrt{n}$-approximation for odd values of d from [56]. We close the chapter with a supplementary note on the treewidth of power graphs obtained through observations related to our previous results. These are also summarized in Table 5.1 below.

	Inapproximability	Approximation
Super-polynomial	$2^{\frac{n^{1-\epsilon}}{\rho d}}$ (66)/ $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ (67)	$O^*((e\rho d)^{\frac{2n}{\rho d}})$ (68)/ $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ (69)
Polynomial	$\Delta^{\lfloor d/2 \rfloor - \epsilon}$ (70)	$O(\Delta^{\lfloor d/2 \rfloor})$ (83)
Bipartite graphs	$n^{1/2-\epsilon}$ [42]	$2\sqrt{n}$ (86)

Table 5.1: A summary of our results (theorem numbers), for even/odd values of d.

[1] Versions of the celebrated *PCP theorem*: intuitively, this refers to the (rather complex) machinery behind the most efficient gap-introducing reductions. See [95] for more information.

5.1 Super-polynomial time

This section concerns itself with running-times that are not restricted to being functions polynomial in the size of the input. We begin with an upper bound on the size of the solution in any connected graph that is then employed in obtaining a simple exact exponential-time algorithm.

Lemma 64. *The maximum size of any d-Scattered Set in a connected graph is* $\left\lfloor \frac{n}{\lfloor d/2 \rfloor} \right\rfloor$.

Proof. Given connected graph $G = (V, E)$, let $S \subseteq V$ be a d-scattered set in G. To each $u \in S$, we will assign all vertices at distance $< \lfloor d/2 \rfloor$: let $M(u) \coloneqq \{u\} \cup \{v \in V | d(u, v) < \lfloor d/2 \rfloor\}$. Our aim is to show that for any $u \in S$, it must be $|M(u)| \geq \lfloor d/2 \rfloor$. In other words, for any vertex u in the solution there must be at least $\lfloor d/2 \rfloor - 1$ distinct vertices that are at distance $< \lfloor d/2 \rfloor$ from u and at distance $\geq \lfloor d/2 \rfloor$ from any vertex $w \in S$. Observe that if for some pair $u, w \in S$, we have $M(u) \cap M(w) \neq \emptyset$, then $d(u, w) < d$, as there exists a vertex at distance $< \lfloor d/2 \rfloor$ from both u, w.

Consider some $u \in S$ and let k be the number of vertex-disjoint paths of length $< \lfloor d/2 \rfloor$ starting from u, i.e. such that no vertices are shared between them. Then it must be $|M(u) \setminus \{u\}| \geq k(\lfloor d/2 \rfloor - 1)$. If $V \subseteq M(u)$, then $OPT_d(G) = |S| = 1$ and the claim trivially holds, so we may assume that there is at least one vertex $z \notin M(u)$. This means $k \geq 1$, since G is connected and there must be (at least) one path from z to u of length $\geq \lfloor d/2 \rfloor$. It is then $|M(u) \setminus \{u\}| + 1 \geq \lfloor d/2 \rfloor - 1 + 1$ giving $|M(u)| \geq \lfloor d/2 \rfloor$.

This means that for each vertex u taken in any solution S there must be at least $\lfloor d/2 \rfloor$ distinct vertices in the graph, i.e. G must contain $|S|$ disjoint subsets of size at least $\lfloor d/2 \rfloor$ and the claim follows. □

Theorem 65. *The d-*Scattered Set *problem can be solved in* $O^*((ed)^{\frac{2n}{d}})$ *time.*

Proof. We simply try all sets of vertices of size at most $\left\lfloor \frac{n}{\lfloor d/2 \rfloor} \right\rfloor$ for feasibility and retain the best one found. By Lemma 64, the optimal solution must be contained therein. The number of sets we examine is $\leq \frac{2n}{d}\binom{n}{\frac{2n}{d}} = O^*((ed)^{\frac{2n}{d}})$, that gives the running-time. □

5.1.1 Inapproximability

We now turn our attention to the problem's hardness of approximation in super-polynomial time. We use Theorem 10 in conjunction with straightforward reductions from Independent Set to d-Scattered Set for the two cases depending on the parity of d.

Theorem 66. *Under the randomized ETH, for any even* $d \geq 4$, $\epsilon > 0$ *and* $\rho \leq (2n/d)^{5/6}$, *no* ρ*-approximation for* d-Scattered Set *can take time* $2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon} d^{1-\epsilon}}\right)} \cdot n^{O(1)}$.

Proof. We suppose the existence of a ρ-approximation algorithm for d-Scattered Set of running-time $2^{\left(\frac{n^{1-\epsilon}}{d^{1-\epsilon}\rho^{1+\epsilon}}\right)} \cdot n^{O(1)}$ for some $\epsilon > 0$ and aim to show this would violate the (randomized) ETH. First let $\epsilon_1 > 0$ be such that $\epsilon > \epsilon_1$ and $\epsilon > \frac{2\epsilon_1}{1-3\epsilon_1}$. We next define $\epsilon_2 = \frac{1}{1-\epsilon_1} - 1$ and $r = \rho^{\frac{1-\epsilon_1}{1-3\epsilon_1}}$. Then, given a formula ϕ of 3SAT on N variables, we use the reduction of Theorem 10 with parameters r and ϵ_2 to build a graph G from ϕ, with $|V(G)| = N^{1+\epsilon_2} r^{1+\epsilon_2}$ and maximum degree r, such that with high probability: if ϕ

is satisfiable then $\alpha(G) \geq N^{1+\epsilon_2} r$; if ϕ is not satisfiable then $\alpha(G) \leq N^{1+\epsilon_2} r^{2\epsilon_2}$. Thus an approximation algorithm with ratio $r^{1-2\epsilon_2}$ would permit us to decide if ϕ is satisfiable. We have that $r^{1-2\epsilon_2} = (\rho^{\frac{1-\epsilon_1}{1-3\epsilon_1}})^{1-2\epsilon_2} = (\rho^{\frac{1-\epsilon_1}{1-3\epsilon_1}})^{3-\frac{2}{1-\epsilon_1}} = \rho^{\frac{3+\frac{2\epsilon_1-2}{1-\epsilon_1}-3\epsilon_1}{1-3\epsilon_1}} = \rho$.

We will construct graph H from G as follows (similarly to Theorem 3.10 in [56], see also Figure 5.1): graph H contains a copy of G and a distinct path of $d/2-1$ edges attached to each vertex of G. Without loss of generality, we may assume that any d-scattered set will prefer selecting an endpoint of these attached paths than some vertex from G, as these selections would exclude strictly fewer vertices from the solution, i.e. any solution can only be improved by exchanging any vertex of G with selecting the (other) endpoint of the path attached to it. A pair of vertices that are endpoints of such paths (and not originally in G) will be at distance $\geq 2(d/2-1)+2 = d$, only if the vertices of G to which they are attached are non-adjacent, i.e. if the shortest path between them is of length at least 2. Thus, d-scattered sets in H are in one-to-one correspondence with independent sets in G and $\alpha(G) = OPT_d(H)$. The size of H is $n = |V(H)| = |V(G)|(d/2-1+1) = N^{1+\epsilon_2} r^{1+\epsilon_2}(d/2)$. Note also that $\rho \leq N^5$ and thus $\rho \leq (2n/d)^{5/6}$.

If ϕ is satisfiable then $OPT_d(H) = \alpha(G) \geq N^{1+\epsilon_2} r$, while if ϕ is not satisfiable then $OPT_d(H) = \alpha(G) \leq N^{1+\epsilon_2} r^{2\epsilon_2}$. Therefore, applying the supposed ρ-approximation for d-Scattered Set on H would permit us to solve 3SAT in time $2^{\left(\frac{n^{1-\epsilon}}{d^{1-\epsilon}\rho^{1+\epsilon}}\right)} \cdot n^{O(1)}$, with high probability. We next show that this would violate the ETH, i.e. $2^{\left(\frac{n^{1-\epsilon}}{d^{1-\epsilon}\rho^{1+\epsilon}}\right)} \cdot n^{O(1)} = 2^{o(N)}$. We have:

$$n = N^{1+\epsilon_2} r^{1+\epsilon_2}(d/2) \Rightarrow N = \left(\frac{2n}{d}\right)^{1-\epsilon_1} \cdot \frac{1}{\rho^{\left(\frac{1-\epsilon_1}{1-3\epsilon_1}\right)}} \tag{5.1}$$

And we then need to show:

$$2^{\left(\frac{n^{1-\epsilon}}{d^{1-\epsilon}\rho^{1+\epsilon}}\right)} \cdot n^{O(1)} = 2^{o(N)} = 2^{o\left(\left(\frac{2n}{d}\right)^{1-\epsilon_1} \cdot \frac{1}{\rho^{\left(\frac{1-\epsilon_1}{1-3\epsilon_1}\right)}}\right)} \tag{5.2}$$

Observe that, since $\epsilon > \epsilon_1$ and $\epsilon > \frac{2\epsilon_1}{1-3\epsilon_1}$, it is:

$$(n/d)^{1-\epsilon} < (2n/d)^{1-\epsilon_1} \tag{5.3}$$

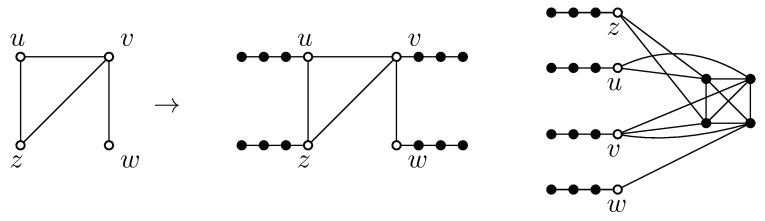
$$(1/\rho^{1+\epsilon}) < \left(1/\rho^{(\frac{1-\epsilon_1}{1-3\epsilon_1})}\right) \tag{5.4}$$

Which then gives:

$$\lim_{(n,\rho)\to\infty} \frac{2^{\left((\frac{n}{d})^{1-\epsilon} \cdot \frac{1}{\rho^{1+\epsilon}}\right)}}{2^{\left((\frac{2n}{d})^{1-\epsilon_1} \cdot \frac{1}{\rho^{\left(\frac{1-\epsilon_1}{1-3\epsilon_1}\right)}}\right)}} = 0 \tag{5.5}$$

□

The following reduction from Independent Set to d-Scattered Set for odd values of d uses a construction that includes a copy of every edge of the original graph (an *edge gadget*, see Figure 5.1). This necessity is responsible for the difference in running-times and is due to the parity idiosyncrasies of the problem as discussed above.



$d = 2$ $d = 8$ $d = 9$

Figure 5.1: Examples of the constructions for even (center) and odd (right) values of d. Note the existence of an edge gadget for the odd case.

Theorem 67. *Under the randomized ETH, for any odd $d \geq 5$, $\epsilon > 0$ and $\rho \leq (2n/d)^{5/6}$, no ρ-approximation for d-Scattered Set can take time $2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon}(d+\rho)^{1+\epsilon}}\right)} \cdot n^{O(1)}$.*

Proof. We suppose the existence of a ρ-approximation algorithm for d-Scattered Set of running-time $2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon}(d+\rho)^{1+\epsilon}}\right)} \cdot n^{O(1)}$ for some $\epsilon > 0$ and aim to show this would violate the (randomized) ETH. We let $\epsilon_1 > 0$ be such that $\epsilon > \frac{\epsilon_1}{1+\epsilon_1}$ and also $\epsilon > \frac{2\epsilon_1}{1-2\epsilon_1}$, as well as $\epsilon > \frac{2\epsilon_1^2+\epsilon_1}{1-2\epsilon_1^2-\epsilon_1}$. We then set $r = \rho^{\left(\frac{1}{1-2\epsilon_1}\right)}$. Given a formula ϕ of 3SAT on N variables, we build graph G from ϕ using the reduction of Theorem 10 with parameters r and ϵ_1. The size of G is $|V(G)| = N^{1+\epsilon_1} r^{1+\epsilon_1}$, its maximum degree is r and with high probability: if ϕ is satisfiable then $\alpha(G) \geq N^{1+\epsilon_1} r$; if ϕ is not satisfiable then $\alpha(G) \leq N^{1+\epsilon_1} r^{2\epsilon_1}$. Therefore an approximation algorithm with ratio $r^{1-2\epsilon_1} = \rho$ would permit us to decide if ϕ is satisfiable.

For odd $d \geq 5$, we will construct graph H from G as follows (again, a similar reduction is alluded to in the proof of Theorem 3.10 from [56] and partly also used for Corollary 1 from [42]): we make a vertex in H for each vertex of G and we also attach a distinct path of $(d-3)/2$ edges to each of them. We then make a vertex for every edge of G, turn all these vertices into a clique and also connect each one to the two vertices of H representing its endpoints. In this way, all pairs of vertices in H are at distance $\leq 2(d-1)/2 + 1 = d$ and the only vertices at exactly this distance are pairs of leaves on paths added to vertices that do not share a common neighbor representing some edge of G. Thus, d-scattered sets in H are again in one-to-one correspondence with independent sets in G and $\alpha(G) = OPT_d(H)$. The size of H is $n = |V(H)| = |V(G)|(d-1)/2 + |E(G)|$. The construction of [26] builds a graph where every vertex has degree at least one and at most r, therefore $|E(G)| \geq |V(G)|$ and $|E(G)| \leq |V(G)|r/2$, that gives $n \leq N^{1+\epsilon_1} r^{1+\epsilon_1} (\frac{d+r-1}{2})$, while $\rho \leq N^5$, with $n \geq N\rho(d+1)/2$, that gives $\rho \leq (2n/d)^{5/6}$.

If ϕ is satisfiable then $OPT_d(H) = \alpha(G) \geq N^{1+\epsilon_1} r$, while if ϕ is not satisfiable then $OPT_d(H) = \alpha(G) \leq N^{1+\epsilon_1} r^{2\epsilon_1}$. Thus the supposed ρ-approximation for d-Scattered Set on H would permit us to solve 3SAT in time $2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon}(d+\rho)^{1+\epsilon}}\right)} \cdot n^{O(1)}$, with high

probability. We next show that this would violate the ETH, i.e. $2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon}(d+\rho)^{1+\epsilon}}\right)}$. $n^{O(1)} = 2^{o(N)}$. It is:

$$n \leq N^{1+\epsilon_1} r^{1+\epsilon_1} \left(\frac{d+r-1}{2}\right) \Rightarrow \tag{5.6}$$

$$\Rightarrow 2^N \geq 2^{\left(\left(\frac{2n}{d+r-1}\right)^{\frac{1}{1+\epsilon_1}} \cdot \frac{1}{r}\right)} = 2^{\left(\left(\frac{2n}{d+\rho^{\left(\frac{1}{1-2\epsilon_1}\right)}-1}\right)^{\frac{1}{1+\epsilon_1}} \cdot \frac{1}{\rho^{\left(\frac{1}{1-2\epsilon_1}\right)}}\right)} \tag{5.7}$$

Observe it is $(d+\rho)^{\frac{1}{1-2\epsilon_1}} > (d+\rho^{\frac{1}{1-2\epsilon_1}}-1)$ and so $2^N > 2^{\left(\left(\frac{2n}{(d+\rho)^{\frac{1}{1-2\epsilon_1}}}\right)^{\frac{1}{1+\epsilon_1}} \cdot \frac{1}{\rho^{\left(\frac{1}{1-2\epsilon_1}\right)}}\right)}$. We thus then require:

$$\lim_{(n,\rho)\to\infty} \frac{2^{\left(\frac{n^{1-\epsilon}}{\rho^{1+\epsilon}(d+\rho)^{1+\epsilon}}\right)}}{2^{\left(\left(\frac{2n}{(d+\rho)^{\frac{1}{1-2\epsilon_1}}}\right)^{\frac{1}{1+\epsilon_1}} \cdot \frac{1}{\rho^{\left(\frac{1}{1-2\epsilon_1}\right)}}\right)}} = 0 \tag{5.8}$$

This is shown by the following inequalities:

$$\epsilon > \frac{\epsilon_1}{1+\epsilon_1} \Rightarrow n^{1-\epsilon} < 2n^{\frac{1}{1+\epsilon_1}} \tag{5.9}$$

$$\epsilon > \frac{2\epsilon_1^2+\epsilon_1}{1-2\epsilon_1^2-\epsilon_1} \Rightarrow \frac{1}{(d+\rho)^{1+\epsilon}} < \frac{1}{(d+\rho)^{\frac{1}{(1+\epsilon_1)(1-2\epsilon_1)}}} \tag{5.10}$$

$$\epsilon > \frac{2\epsilon_1}{1-2\epsilon_1} \Rightarrow \frac{1}{\rho^{1+\epsilon}} < \frac{1}{\rho^{(\frac{1}{1-2\epsilon_1})}} \tag{5.11}$$

□

5.1.2 Approximation

We complement the above hardness results with approximation algorithms of almost matching super-polynomial running-times. Similarly to the exact algorithm of Theorem 65, the upper bound from the beginning of this section is used for even values of d, while for the odd values this idea is combined with a greedy scheme based on minimum vertex degree.

Theorem 68. *For any even $d \geq 2$ and any $\rho \leq \frac{n}{\lfloor d/2 \rfloor}$, there is a ρ-approximation algorithm for d-*Scattered Set *of running-time $O^*((e\rho d)^{\frac{2n}{\rho d}})$.*

Proof. From Lemma 64 we know that the maximum size of a d-scattered set is $\lfloor \frac{n}{\lfloor d/2 \rfloor} \rfloor$. We thus simply try all sets of vertices of size at most $\frac{n}{\rho\lfloor d/2 \rfloor}$ for feasibility and retain the best one: these are $\leq \frac{n}{\rho\lfloor d/2 \rfloor}(\frac{n}{\frac{n}{\rho\lfloor d/2 \rfloor}}) = O^*((e\rho d)^{\frac{2n}{\rho d}})$, that gives the running-time. If the graph is not connected, we can apply Lemma 64 to each connected component C of size n_C and then consider all subsets of size at most $\frac{n_C}{\rho\lfloor d/2 \rfloor}$ in each C. □

Theorem 69. *For any odd $d \geq 3$ and any $\rho \leq \frac{n}{\lfloor d/2 \rfloor}$, there is a ρ-approximation algorithm for d-Scattered Set of running-time $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$.*

Proof. Let $q = (d-1)/2$ and G' be the q-th power of graph G. We then claim that any d-scattered set S in G is a 3-scattered set in G' and vice-versa: if S is a d-scattered set in G, then for any pair $u, v \in S$, it is $d_G(u,v) \geq d$. For some pair of distinct vertices w, z on a shortest path between u, v in G, it must be $d_G(u,w) = d_G(z,v) = q$, while also $d_G(u,z) = d_G(w,v) > q$, i.e. w and z are two vertices on a shortest path from u to v, each at equal distance q from their closest endpoint (v or u). Then $d_{G'}(u,w) = d_{G'}(z,v) = 1$. Since w, z are distinct, it must be $d_{G'}(w,z) \geq 1$ which gives also $d_{G'}(u,v) \geq 3$, since $d_{G'}(u,z) = d_{G'}(w,v) > 1$.

If S is a 3-scattered set in G', then for any pair $u, v \in S$ it is $d_{G'}(u,v) \geq 3$. Now, any shortest path in G between u, v must contain two distinct vertices w, z for which $d_G(u,w) = q$ and $d_G(z,v) = q$, while $d_G(u,z) = d_G(w,v) > q$. If no such pair of vertices exists in G, then $d_{G'}(u,v) < 3$: any pair of vertices at distance $\leq q$ in G are adjacent in G' and so for the distance between u, v in G' to be at least 3, there must be two vertices each at distance $\geq q$ from u, v in G. From this we get that $d_G(u,v) \geq 2q + 1 = d$, since $d_G(w,z) \geq 1$, as $w \neq z$.

The algorithm then proceeds in two phases. For the first phase, so long as there exists an unmarked vertex v_i in G' of minimum degree $< \rho$, we mark v_i as 'selected' and add it to $S_1 \subseteq V$, marking all vertices at distance ≤ 2 from v_i in G' as 'excluded' and adding them to $X_i \subseteq V$. That is, $X_i = N^2_{G'}(v_i)$ and we let $X = N^2_{G'}(S_1)$, i.e. $X = X_1 \cup \cdots \cup X_{|S_1|}$. We also let the remaining (unmarked) vertices belong to $H \subseteq V$. Thus when this procedure terminates we have V partitioned into three sets S_1, X, H, while the degree of any vertex in H is $\geq \rho$. For the second phase, we try all subsets of vertices of H of size at most $\frac{2n}{\rho(\rho+\lfloor d/2 \rfloor)}$ for feasibility and retain the best one. These are $\leq \frac{2n}{\rho(\rho+\lfloor d/2 \rfloor)}\binom{n}{\frac{2n}{\rho(\rho+\lfloor d/2 \rfloor)}} = O^*((e\rho d)^{2n/\rho(d+\rho)})$, giving the upper bound on the running-time.

Now let S_1^* be a 3-scattered set of maximum size in the subgraph of G' induced by $S_1 \cup X$, i.e. $|S_1^*| = OPT_3(G'[S_1 \cup X])$ and S_2^* be a 3-scattered set of maximum size in the subgraph of G' induced by H, i.e. $|S_2^*| = OPT_3(G'[H])$. As the degree of any vertex $v_i \in S_1$ is $< \rho$, we have (1): $|S_1^*| < \rho|S_1|$, since for every vertex u in $N^1_{G'}(v_i)$, for $v_i \in S_1$, 3-scattered set S_1^* can contain at most one vertex w from $N^1_{G'}(u)$, as the distance between w and another neighbor of u is ≤ 2. For the second phase, it must be $|S_2^*| \leq n/\rho \Rightarrow 1/|S_2^*| \geq \rho/n$, since all vertices of H are of degree $\geq \rho$ and these neighborhoods are disjoint: if two vertices of S_2^* share a common neighbor then they cannot belong in a 3-scattered set. From Lemma 64, we also know that $|S_2^*| \leq n/\lfloor d/2 \rfloor \Rightarrow 1/|S_2^*| \geq \lfloor d/2 \rfloor/n$. Adding the two inequalities gives $2/|S_2^*| \geq \frac{\rho+\lfloor d/2 \rfloor}{n} \Rightarrow |S_2^*| \leq \frac{2n}{\rho+\lfloor d/2 \rfloor}$. Furthermore, it is $|S_2| \leq \frac{2n}{\rho(\rho+\lfloor d/2 \rfloor)}$, by construction. Dividing the two inequalities gives (2): $\frac{|S_2^*|}{|S_2|} \leq \rho \Rightarrow |S_2^*| \leq \rho|S_2|$. From (1) and (2) we get that $|S_1^*| + |S_2^*| \leq \rho(|S_1| + |S_2|)$. It is $OPT_d(G) = OPT_3(G') \leq |S_1^*| + |S_2^*|$ since $S_1 \cup X$ and H form a partition of G'. Our algorithm returns a solution of size $|S_1| + |S_2|$ and thus our approximation ratio is $\frac{OPT_d(G)}{|S_1|+|S_2|} \leq \frac{|S_1^*|+|S_2^*|}{|S_1|+|S_2|} \leq \rho$. If the graph is not connected, we can apply Lemma 64 to each connected component C of size n_C and then try all subsets of size at most $\frac{n_C}{\rho\lfloor d/2 \rfloor}$ in each C and obtain an additive version of (2) for each component. □

5.2 Polynomial Time

We now focus on the behaviour of the problem in terms of strictly polynomial-time approximation. We first examine graphs of bounded degree and provide a tight bound on the achievable approximation ratio, before turning to bipartite graphs in order to finalize the classification in terms of approximability by considering the only open remaining case (when d is even).

5.2.1 Inapproximability

Here we show that for sufficiently large Δ and any $\epsilon_1 > 0, d \geq 4$, the d-Scattered Set problem is inapproximable to $\Delta^{d/2-\epsilon_1}$ on graphs of degree bounded by Δ, unless NP$\subseteq$BPP. Let us first summarize our reduction. Starting from an instance of Independent Set of bounded degree, we create an instance of d-Scattered Set where the degree is (roughly) the $d/2$-th square root of that of the original instance. As we are able to maintain a direct correspondence of solutions in both instances, the $\Delta^{1-\epsilon'}$-inapproximability of IS implies the $\Delta^{d/2-\epsilon_1}$-inapproximability of d-Scattered Set.

The technical part of our reduction involves preserving the adjacency between vertices of the original graph without increasing the maximum degree (too far) beyond $\Delta^{2/d}$. We are able to construct a regular tree as a gadget for each vertex and let the edges of the leaves (their total number being equal to Δ) represent the edges of the original graph. To ensure that our gadget has some useful properties (i.e. small diameter), we overlay a number of extra edges over each level of the tree (i.e. between vertices at equal distance from the root), only sacrificing a small increase in maximum degree. Our complexity assumption is NP$\not\subseteq$BPP, since for the $\Delta^{1-\epsilon'}$-inapproximability of IS we use the randomized reduction from SAT of [26] (Theorem 10 above). In particular, we will prove the following theorem:

Theorem 70. *For sufficiently large Δ and any $d \geq 4, \epsilon \in (0, \lfloor d/2 \rfloor)$, there is no polynomial-time approximation algorithm for d-Scattered Set with ratio $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ for graphs of maximum degree Δ, unless NP$\subseteq$BPP.*

Construction: Let $\delta = \left\lceil \sqrt[\lfloor d/2 \rfloor]{\Delta} \right\rceil$. Given $\epsilon_1 \in (0, \lfloor d/2 \rfloor)$ and an instance of Independent Set $G = (V, E)$, where the degree of any vertex is bounded by Δ, we will construct an instance $G' = (V', E')$ of d-Scattered Set, where the degree is bounded[2] by $\delta^{1+\epsilon_2} = 6\delta^{1+2\epsilon_1/d}$, for $\epsilon_2 = 2\epsilon_1/d + \log_\delta 3 > 2\epsilon_1/d$, while $OPT_2(G) = OPT_d(G')$. We assume Δ is sufficiently large for $\epsilon_1 \geq \frac{d(\log(\log(\Delta))+c)}{4\log(\Delta)/d}$, where $c \leq 10$ is a small constant, for reasons that become apparent in the following.

Our construction for G' builds a gadget $T(v)$ for each vertex $v \in V$. For even d, each gadget $T(v)$ is composed of a $(\delta+1)$-regular tree of height $d/2-1$ and we refer to vertices of $T(v)$ at distance exactly i from the root t_v as being in the i-th *height-level* of $T(v)$, letting each such subset be denoted by $T_i(v)$. That is, every vertex of $T_i(v)$ has one neighbor in $T_{i-1}(v)$ (its parent) and δ neighbors in $T_{i+1}(v)$ (its children). For odd values of d, the difference is in the height of each tree being $\lfloor d/2 \rfloor$ instead of $d/2-1$.

Since for even d the number of leaves of $T(v)$ is $\delta^{d/2-1} = (\Delta^{2/d})^{d/2-1} = \Delta^{1-2/d}$ and each such leaf also has $\delta = \Delta^{2/d}$ edges, the number of edges leading outside each gadget is

[2]We note that this value of ϵ_2 is for odd values of d. For d even, the correct value is such that we have the (slightly lower) bound $\delta^{1+\epsilon_2} = \delta + 3\delta^{1+2\epsilon_1/d}$, but we write ϵ_2 for both cases to simplify notation.

$\delta^{d/2} = \Delta$ and we let each of them correspond to one edge of the original vertex v in G, i.e. we add an edge between a leaf x_v of $T(v)$ and a leaf y_u of $T(u)$, if $(v,u) \in E$. For odd d, the number of leaves is $\left(\left\lceil \sqrt[\lfloor d/2 \rfloor]{\Delta} \right\rceil\right)^{\lfloor d/2 \rfloor}$ (i.e. at least Δ) and we let each leaf correspond to an edge of the original vertex v in G, i.e. we *identify* two such leaves x_v, y_u of two gadgets $T(v), T(u)$, if $(v,u) \in E$ in G. In this way, the gadgets $T(v), T(u)$ share a common "leaf" of degree 2, that is at distance $\lfloor d/2 \rfloor$ from both roots $t_v \in T(v), t_u \in T(u)$. See Figure 5.2 for an illustration.

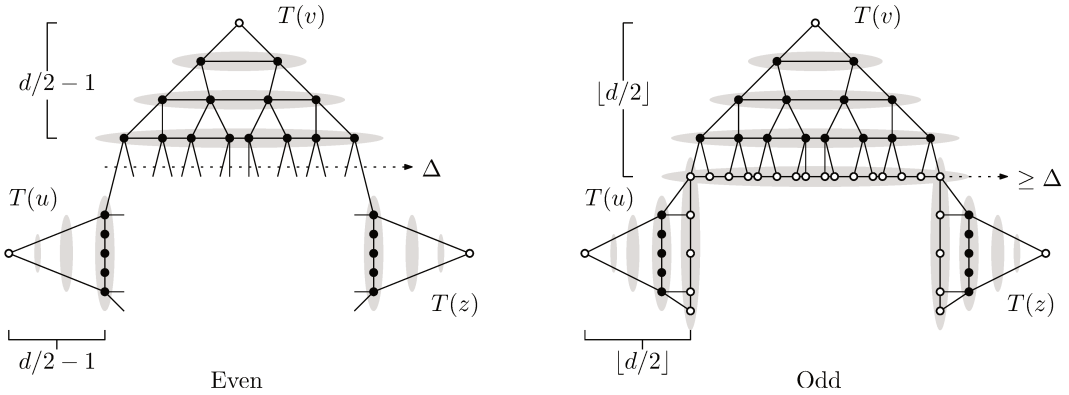


Figure 5.2: Examples of our constructions for a path on three vertices (u, v, z) and even/odd d. Ellipses in grey designate the overlaid edges on each height-level.

Next, in order to make the diameter of our gadgets at most equal to their height, we will add a number of edges between the vertices of each height-level i of each gadget $T(v)$, for every $v \in V$. We will first add the edges of a cycle plus a random matching (using a technique from [17]) and then the edges of an appropriately chosen power graph of this subgraph containing the edges of the cycle plus the matching. These edges will be overlaid on each height-level, meaning our final construction will contain the edges of the tree, the cycle, the matching, as well as the power graph.

For even d and each gadget $T(v)$, we first make all vertices $T_i(v)$ at each height-level $i < 1 + \epsilon_2$ into a clique. For larger height-levels $i \in [1 + \epsilon_2, d/2 - 1]$, we first make the vertices into a cycle (arbitrarily ordered) and then also add a random matching, i.e. the edges between each pair of a random partition of $T_i(v)$ into disjoint pairs (plus a singleton if $|T_i(v)|$ is odd). Letting $P_i(v)$ denote these edges of the cycle plus the matching for each $T_i(v)$, we define the subgraph $H_i(v) = (T_i(v), P_i(v))$ and compute the $\lceil((1 + 2\epsilon_1/d)\log_2(\delta))\rceil$-power graph of $H_i(v)$, finally also adding its edges to G'. For odd d and each gadget $T(v)$, we again make the vertices $T_i(v)$ at each height-level $i < 1 + \epsilon_2$ into a clique and for larger height-levels $i \in [1 + \epsilon_2, \lfloor d/2 \rfloor]$, we follow the same process.

This concludes our construction, while to prove our claims on the diameter of our gadgets we also make use of the following statements:

Theorem 71 ([17], Theorem 1)**.** *Let G be a graph formed by adding a random matching to an n-cycle. Then with probability tending to 1 as n goes to infinity, G has diameter upper-bounded by $\log_2(n) + \log_2(\log(n)) + c$, where c is a small constant (at most 10).*

Lemma 72. *Let G be a graph of diameter $\leq a$. Then the diameter of the b-power graph G^b is $\leq \lceil a/b \rceil$, for any integer $b < a$.*

Proof. Consider a path P of (maximum) length $\leq a$ between two vertices v, u in G. Taking the b-power of G adds all edges between vertices of P at distance $\leq b$. This means vertex v will be adjacent in G^b to a vertex x_1 on P that was at distance b from u in G. This vertex x_1 will be in turn adjacent to another vertex x_2 on P that was at distance $2b$ from v in G. Carrying on like this we can find a sequence $x_1, \ldots, x_i$ of vertices of P, each at distance b in G from its predecessor and follower in the sequence, that form a path P' in G^b. Since the length of P is at most a, the maximum number i of vertices in the sequence until we reach u is $\lceil a/b \rceil$, giving the length of P' in G^b. □

We are now ready to argue about the maximum degree of any vertex in our construction.

Lemma 73. *The maximum degree of any vertex in G' is $\leq \delta + 3\delta^{1+2\epsilon_1/d}$ for even d and $\leq 6\delta^{1+2\epsilon_1/d}$ for odd d.*

Proof. Observe that for d even, the degree of any vertex is bounded by the sum of the $\delta+1$ edges of the tree plus the number of edges added by the power graph (including the three edges of the cycle and matching): $\sum_{k=0}^{\lceil (1+2\epsilon_1/d)\log_2(\delta)\rceil}(3 \cdot 2^k) = 3 \cdot 2^{\lceil (1+2\epsilon_1/d)\log_2(\delta)-1\rceil} - 1 \leq 3 \cdot 2^{(1+2\epsilon_1/d)\log_2(\delta)} - 1 = 3 \cdot \delta^{1+2\epsilon_1/d} - 1$, for a total of $\delta + 3\delta^{1+2\epsilon_1/d}$.

For d odd, we note that the degree of all other vertices is strictly lower than that of the "shared" leaves between gadgets, since each leaf between two gadgets $T(v), T(u)$ (representing the edge (v, u) of G) will belong to two subgraphs $H_{\lfloor d/2 \rfloor}(v)$ and $H_{\lfloor d/2 \rfloor}(u)$. Thus their degree will be $2 + 2(3 \cdot \delta^{1+2\epsilon_1/d} - 1) = 6\delta^{1+2\epsilon_1/d}$. □

We then bound the diameter of our gadgets in order to guarantee that the solutions in our reduction will be well-formed. Our statement is probabilistic and conditional on our assumption on the size of Δ as being sufficiently large.

Lemma 74. *With high probability, the diameter of each gadget $T(v)$ is $d/2 - 1$ for even d and $\lfloor d/2 \rfloor$ for odd d, for sufficiently large Δ.*

Proof. First, observe that for sufficiently large n, $c \leq 10$ and $\epsilon_1 \in (0, \lfloor d/2 \rfloor)$, it is $\log_2(\log(n)) + c < (2\epsilon_1/d)\log_2(n)$. For even d, our construction uses n-cycles of length $n = \delta^i$ for each $i \in [1 + \epsilon_2, d/2 - 1]$, meaning that Δ must be sufficiently large for $\epsilon_1 \geq \frac{d(\log(2i\log(\Delta)/d)+c)}{4i\log(\Delta)/d}$, while for odd d it is $i \in [1 + \epsilon_2, \lfloor d/2 \rfloor]$. As noted above, our assumption for Δ requires that it is sufficiently large for $\epsilon_1 \geq \frac{d(\log(\log(\Delta))+c)}{4\log(\Delta)/d}$, which is $> \frac{d(\log(2i\log(\Delta)/d)+c)}{4i\log(\Delta)/d}$ for the required range of i in both cases.

By Theorem 71, the distance between any pair of vertices at height-level i after adding the edges of $P_i(v)$ is at most $\log_2(\delta^i) + \log_2(\log(\delta^i)) + c$ (with high probability). This is $< (1+2\epsilon_1/d)\log_2(\delta^i)$ for sufficiently large Δ. By Lemma 72, taking the $\lceil((1+2\epsilon_1/d)\log_2(\delta))\rceil$-power of $H_i(v)$ shortens the distance to at most $\frac{(1+2\epsilon_1/d)\log_2(\delta^i)}{\lceil(1+2\epsilon_1/d)\log_2(\delta)\rceil} \leq i$, for each height-level $i \in [1 + \epsilon_2, d/2 - 1]$. For smaller values of i, the vertices of each height-level form a clique and the distance between any pair of them is thus at most 1.

For odd values of d, the size n of the cycles we use is again δ^i, with $i \in [1 + \epsilon_2, \lfloor d/2 \rfloor]$ and we thus have once more that for sufficiently large Δ the distance between any pair of vertices after adding the edges of $P_i(v)$ to each height-level i of each gadget $T(v)$ is at most $(1 + 2\epsilon_1/d)\log_2(\delta^i)$ (with high probability) and at most i after taking the $\lceil((1 + 2\epsilon_1/d)\log_2(\delta))\rceil$-power of $H_i(v)$. Again, for smaller $i < 1 + \epsilon_2$, $T_i(v)$ is a clique.

Since at each height-level i, no pair of vertices is at distance $> i$ with $i \leq d/2 - 1$ for even d and $i \leq \lfloor d/2 \rfloor$ for odd d, the distance between any vertex x at some height-level i_x to another vertex y at height-level $i_y > i_x$ will be at most i_x from x to the root of the subtree of $T(v)$ (at level i_x) that contains y. From there to y it will be at most $d/2 - 1 - i_x$ for even d and at most $\lfloor d/2 \rfloor - i_x$ for odd d. Furthermore, the distance from the root of $T(v)$ to a leaf is exactly $d/2 - 1$ for even d and exactly $\lfloor d/2 \rfloor$ for odd d. □

We finalize our argument with a series of lemmas leading to the proof of Theorem 70, that detail the behaviour of solutions that can form in our construction, relative to the independence of vertices in the original graph.

Lemma 75. *No d-Scattered Set in G' can contain a vertex from gadget $T(v)$ and a vertex from gadget $T(u)$, if $(u, v) \in E$.*

Proof. Since $(u, v) \in E$, there is an edge $(x_v, y_u) \in E'$ between a leaf $x_v \in T(v)$ and $y_u \in T(u)$ for even d, while for for odd d the leaf x belongs to both $T(v), T(u)$ and is at distance $\lfloor d/2 \rfloor$ from each of their roots. Thus for even d the maximum distance from any vertex of $T(v)$ to $y_u \in T(u)$ is $d/2 - 1 + 1 = d/2$, by Lemma 74, and for odd d this is $\lfloor d/2 \rfloor$. Since, by the same lemma, the diameter of $T(u)$ is $d/2 - 1$ for even d and $\lfloor d/2 \rfloor$ for odd d, there is no vertex of $T(u)$ that can be in any d-Scattered Set along with any vertex of $T(v)$, as the maximum distance is $\leq d/2 + d/2 - 1 = d - 1$ for even d and $\leq \lfloor d/2 \rfloor + \lfloor d/2 \rfloor = d - 1$ for odd d. □

Lemma 76. *If $(u, v) \notin E$, then the distance between the root t_v of $T(v)$ and the root t_u of $T(u)$ is at least d.*

Proof. Since $(u, v) \notin E$, then there is no edge between any pair of leaves x_u of $T(u)$ and y_v of $T(v)$ for even d. Thus the shortest possible distance between any such pair of leaves is 2 for even d, through a third leaf z_w of another gadget $T(w)$ corresponding to a vertex w adjacent to both u and v in G. The distance from t_v to any leaf of $T(v)$ is $d/2 - 1$ and the distance from t_u to any leaf of $T(u)$ is also $d/2 - 1$. Thus the distance from t_u to t_v must be at least $d/2 - 1 + d/2 - 1 + 2 = d$.

For odd d, there is no shared leaf x between the two gadgets, i.e. at distance $\lfloor d/2 \rfloor$ from both roots. Thus the distance between two leaves $x_u \in T(u)$ and $y_v \in T(v)$ is at least 1, if each of these is shared with a third gadget $T(w)$ corresponding to a vertex w that is adjacent to both u and v in G. The distance from t_v to any leaf of $T(v)$ is $\lfloor d/2 \rfloor$ and the distance from t_u to any leaf of $T(u)$ is also $\lfloor d/2 \rfloor$. Thus the distance from t_u to t_v is at least $2\lfloor d/2 \rfloor + 1 = d$. □

Lemma 77. *For any independent set S in G, there is a d-Scattered Set K in G', with $|S| = |K|$.*

Proof. Given an independent set S in G, we let K include the root vertex $t_v \in T(v)$ for each $v \in S$. Clearly, $|S| = |K|$. Since S is independent, there is no edge (u, v) between any pair $u, v \in S$ and thus, by Lemma 76, vertices t_v and t_u are at distance at least d. □

Lemma 78. *For any d-Scattered Set K in G', there is an independent set S in G, with $|K| = |S|$.*

Proof. Given a d-Scattered Set K in G, we know there is at most one vertex from each gadget $T(v)$ in K, since its diameter is $d/2-1$ for even d and $\lfloor d/2 \rfloor$ for odd d, by Lemma 74. Furthermore, for any two vertices $x, y \in K$, we know by Lemma 75 that if $x \in T(u)$ and $y \in T(v)$ for gadgets corresponding to vertices $u, v \in V$, then $(u, v) \notin E$ and thus u, v are independent in G. We let set S contain each vertex $v \in V$ whose corresponding gadget $T(v)$ contains a vertex of K. These vertices are all independent and also $|K| = |S|$. □

Proof of Theorem 70. We suppose the existence of a polynomial-time approximation algorithm for d-Scattered Set with ratio $\Delta^{\lfloor d/2 \rfloor - \epsilon_1}$ for graphs of maximum degree Δ and some $0 < \epsilon_1 < d/2$. We assume Δ is sufficiently large for $\epsilon_1 \geq \frac{d(\log(\log(\Delta))+10)}{4\log(\Delta)/d}$.

Starting from a formula ϕ of SAT on N variables, where N is also sufficiently large, i.e $N > \Delta^{1/(5+O(\epsilon'))}$ (where ϵ' is defined below), we use Theorem 10 to produce an instance $G = (V, E)$ of Independent Set on $|V| = N^{1+\epsilon'}\Delta^{1+\epsilon'}$ vertices and of maximum degree Δ, such that with high probability: if ϕ is satisfiable, then $\alpha(G) \geq N^{1+\epsilon'}\Delta$; if ϕ is not satisfiable, then $\alpha(G) \leq N^{1+\epsilon'}\Delta^{2\epsilon'}$. Thus approximating Independent Set in polynomial time on G within a factor of $\Delta^{1-2\epsilon'}$, for $\epsilon' > 0$, would permit us to decide if ϕ is satisfiable, with high probability.

We next use the above construction to create an instance G' of d-Scattered Set where the degree is bounded by $6\delta^{1+2\epsilon_1/d} = \delta^{1+\epsilon_2}$, for $\epsilon_2 > 2\epsilon_1/d$, by Lemma 73. Slightly overloading notation, we let $\epsilon_3 \geq \epsilon_2$ be such that $\delta^{1+\epsilon_2} = (\lceil \Delta^{\frac{1}{\lfloor d/2 \rfloor}} \rceil)^{1+\epsilon_2} = (\Delta^{\frac{1}{\lfloor d/2 \rfloor}})^{1+\epsilon_3}$. We now let $\epsilon' = \frac{\epsilon_1(1+\epsilon_3)-\epsilon_3\lfloor d/2 \rfloor}{2\lfloor d/2 \rfloor}$. Note that $\epsilon' > 0$, since $\epsilon_3 \geq \epsilon_2 > 2\epsilon_1/d$.

We then apply the supposed approximation for d-Scattered Set on G'. This returns a solution at most $(\delta^{1+\epsilon_2})^{\lfloor d/2 \rfloor - \epsilon_1} = \Delta^{(1-\frac{\epsilon_1}{\lfloor d/2 \rfloor})(1+\epsilon_3)} = \Delta^{1-\frac{\epsilon_1(1+\epsilon_3)-\epsilon_3\lfloor d/2 \rfloor}{\lfloor d/2 \rfloor}} = \Delta^{1-2\epsilon'}$ from the optimum. By Lemma 78 we can find a solution for Independent Set in G of the same size, i.e. we can approximate $\alpha(G)$ within a factor of $\Delta^{1-2\epsilon'}$, again, with high probability (as Lemma 74 is also randomized). This would allow us to decide if ϕ is satisfiable and thus solve SAT in polynomial time with two-sided bounded errors, implying NP⊆BPP. □

5.2.2 Approximation

We next show that any (degree-based) greedy polynomial-time approximation algorithm for d-Scattered Set achieves a ratio of $O(\Delta^{\lfloor d/2 \rfloor})$, thus improving upon the analysis of [43] and the $O(\Delta^{d-1})$- and $O(\Delta^{d-2}/d)$-approximations given therein.

Our strategy is to bound the size of the largest d-scattered set in any graph of maximum degree at most Δ and radius at most $d-1$, centered on some vertex v. The idea is that in one of its iterations our greedy algorithm would select v and thus exclude all other vertices within distance $d-1$ from v, yet an upper bound on the size of the largest possible d-scattered set can guarantee that the ratio of our algorithm will not be too large.

The following definition of our "merge" operation (see also Figure 5.3) will allow us to consider all possible graphs of a given radius and degree and provide upper bounds on the size of the optimal solution in such graphs. These bounds on the size of the optimal are then used to compare it to those solutions produced by our greedy scheme.

Definition 79 (Merge operation)**.** *For two connected graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the* merged *graph $M_{G_1}^{G_2}(v_1, v_2, \mathbf{U}, \mathbf{W})$, where $\mathbf{U} = [u_1, \ldots, u_{k_1}]$, $\mathbf{W} = [w_1, \ldots, w_{k_2}]$ are ordered (possibly empty and with repetitions allowed) sequences of vertices from V_1 and V_2, respectively, is defined as the graph $G' = (V', E')$ obtained by:*

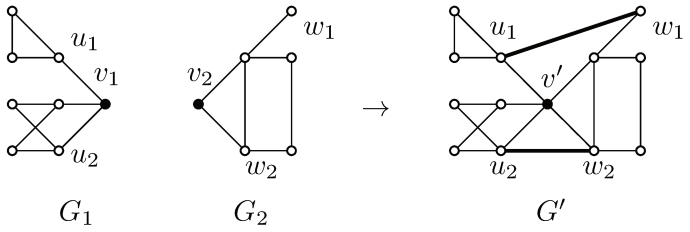


Figure 5.3: An example graph $G' = M_{G_1}^{G_2}(v_1, v_2, [u_1, u_2], [w_1, w_2])$ for G_1, G_2 shown on the left, with edges added by the third merge operation shown in bold.

1. *Identification of vertex $v_1 \in G_1$ and vertex $v_2 \in G_2$, i.e. V' is composed of the union of V_1, V_2 after removal of vertices v_1, v_2 and addition of a new vertex v'.*

2. *Replacement of all edges of v_1, v_2 by new edges with v' as the new endpoint, i.e. E' contains all edges of E_1, E_2, where any edges incident on v_1 or v_2 are now incident on v'.*

3. *Addition of a number of edges between vertices of G_1, G_2, i.e. E' also contains one edge between every pair (u_i, w_j) from $\mathbf{U}, \mathbf{W}$, for $i = j$.*

Lemma 80. *The maximum size of a d-scattered set in any graph of maximum degree at most Δ and radius at most $\lceil d/2 \rceil$ centered on some vertex v is at most Δ.*

Proof. Consider a graph of maximum degree Δ and radius $\lceil d/2 \rceil$ centered on some vertex v: the only pairs of vertices at distance d from each other must be at distance $\geq \lceil d/2 \rceil$ from v (or $\lfloor d/2 \rfloor$ in one side for odd d), as any vertex u at distance $< \lfloor d/2 \rfloor$ from v will be at distance $< d$ from any other vertex z in the graph, since z is at distance $\leq \lceil d/2 \rceil$ from v (due to the graph's radius). Furthermore, for every vertex in the d-scattered set, there must be an edge-disjoint path of length at least $\lceil d/2 \rceil$ to v that is not shared with any other such vertex, i.e. these paths can only share vertex v at distance $\lceil d/2 \rceil$ from their endpoints (the vertices that can be in a d-scattered set). As the degree of v is bounded by Δ, the number of such disjoint paths also cannot be more than Δ. □

Lemma 81. *Given two graphs G_1, G_2, for $G' = M_{G_1}^{G_2}(v_1, v_2, \mathbf{U}, \mathbf{W})$ and any $\mathbf{U}, \mathbf{W}$, it is $OPT_d(G') \leq OPT_d(G_1) + OPT_d(G_2)$.*

Proof. Assume for the sake of contradiction that $OPT_d(G') > OPT_d(G_1) + OPT_d(G_2)$ and let S denote an optimum d-scattered set in G' of this size, with $S_1 = S \cap \{V_1 \setminus \{v_1\} \cup \{v'\}\}$ and $S_2 = S \cap \{V_2 \setminus \{v_2\} \cup \{v'\}\}$ denoting the parts in G_1, G_2, respectively. Since $|S| > OPT_d(G_1) + OPT_d(G_2)$, then for any pair of optimal $K_1 \subseteq V_1$, $K_2 \subseteq V_2$ in G_1, G_2, there must be at least one vertex s in S for which $s \notin K_1$ and $s \notin K_2$, but it must be $s \in S_1$ or $s \in S_2$ (or both, if $s = v'$).

Observe that the distance between any pair of vertices in G_1 (the same holds for G_2) cannot increase in G' after the merge operation, since identification of a pair of vertices between two graphs and addition of any number of edges between the two can only decrease their distance Thus if $s \in V_1$, then S_1 is also a d-scattered set in G_1 (potentially substituting v' for v_1) and so, if $|S_1| > |K_1|$ then K_1 was not optimal for G_1. If $|S_1| \leq |K_1|$, it must be $|S_2| > |K_2|$ contradicting the optimality of K_2 for G_2. Similarly, if $u \in V_2$ we have either $|S_1| > |K_1|$ or $|S_2| > |K_2|$.

If u is the merged vertex v' then there must be at least two other vertices added from V_1, V_2 for $|S| > |K_1| + |K_2|$, since S can only contain v' in the place of $v_1 \in K_1$ and $v_2 \in K_2$. In this case the same argument as above gives the contradiction. □

Lemma 82. *For any graph $G = (V, E)$ of maximum degree at most Δ and radius at most $d-1$ centered on some vertex v, it is $OPT_d(G) \leq O(\Delta^{\lfloor d/2 \rfloor})$.*

Proof. Any graph G of maximum degree at most Δ and radius at most $d-1$ centered on a vertex v can be obtained by the following process: we begin with a graph H of radius at most $\lfloor d/2 \rfloor - 1$ and maximum degree Δ. Let $\{v_1, \ldots, v_k\} \in H$ be the set of vertices at maximum distance from v, i.e. $d_H(v, v_i) = \lfloor d/2 \rfloor - 1$. Since the degree of H is bounded by Δ, it must be $k \leq \Delta^{\lfloor d/2 \rfloor - 1}$. We now let H_i, for each $i \leq k$, denote a series of at most k graphs of radius at most $\lceil d/2 \rceil$ centered on a vertex v_i and maximum degree Δ.

Repeatedly applying the merge operation $M_H^{H_i}(v_1, v_i, \mathbf{U}, \mathbf{W})$ between graph H (or the result of the previous operation) and such a graph H_i we can obtain any graph G of radius at most $d-1$: identifying a vertex $v_j \in H$ (for $j \in [1, k]$) at maximum distance from v with the central vertex v_i of H_i and then adding any number of edges between the vertices of H and H_i (while respecting the maximum degree of Δ), we can produce any graph of radius $\leq d-1$, since the distance from v to each v_j is at most $\lfloor d/2 \rfloor - 1$ and from there to any vertex of H_i it is at most $\lceil d/2 \rceil$. The remaining structure of G can be constructed by the chosen structures of the graphs H, H_i and the added edges between them, i.e. the sequences $\mathbf{U}, \mathbf{W}$.

By Lemma 80 it is $OPT_d(H_i) \leq \Delta$ and by Lemma 81, it must be $OPT_d(G) \leq OPT_d(H) + \sum_{i=1}^{k} OPT_d(H_i) \leq 1 + \Delta \cdot \Delta^{\lfloor d/2 \rfloor - 1} \leq 1 + \Delta^{\lfloor d/2 \rfloor}$. □

Theorem 83. *Any degree-based greedy approximation algorithm for d-Scattered Set achieves a ratio of $O(\Delta^{\lfloor d/2 \rfloor})$ on graphs of degree bounded by Δ.*

Proof. Let $G = (V, E)$ be the input graph and consider the process of our supposed greedy algorithm: it picks a vertex v_i, removes it from consideration along with the set $V_i \subseteq V$ of vertices at distance at most $d-1$ from v_i and continues the process until there are no vertices left to consider. The sets $V_1, \ldots, V_{ALG}$ thus form a partition of G. By Lemma 82, the optimum size of a d-scattered set in any such V_i is at most $O(\Delta^{\lfloor d/2 \rfloor})$ and thus $OPT_d(G) \leq ALG \cdot O(\Delta^{\lfloor d/2 \rfloor})$, by Lemma 81, since G can be seen as the merged graph of $G[V_1], \ldots, G[V_{ALG}]$. □

5.2.3 Bipartite graphs

Finally, we consider bipartite graphs and show that d-Scattered Set is approximable to $2\sqrt{n}$ in polynomial time also for even values of d. Our algorithm will be applied on both sides of the bipartition and each time will only consider vertices from one side as candidates for inclusion in the solution. Appropriate sub-instances of Set Packing are then defined and solved using the known $\sqrt{n}$-approximation for that problem.

Definition 84. *For a bipartite graph $G = (A \cup B, E)$, let $1OPT_d(G)$ denote the size of the largest* one-sided *d-scattered set of G, i.e. a set that only includes vertices from the same side of the bipartition A or B, but not both.*

Lemma 85. *For a bipartite graph $G = (A \cup B, E)$, it is $1OPT_d(G) \geq OPT_d(G)/2$.*

Proof. Consider an optimal solution $S \subseteq A \cup B$ with $|S| = OPT_d(G)$. Then at least half of the vertices of S are contained in one side of G, i.e. it is either $|S \cap A| \geq |S|/2$ or $|S \cap B| \geq |S|/2$ (or both if $|S \cap A| = |S \cap B| = |S|/2$). By definition, it is also $1OPT_d(G) \geq |S \cap A|$ and $1OPT_d(G) \geq |S \cap B|$. Thus in both cases it must be $1OPT_d(G) \geq OPT_d(G)/2$. □

Theorem 86. *For any bipartite graph $G = (A \cup B, E)$ of size n and d even, the d-*Scattered Set *problem can be approximated within a factor of $2\sqrt{n}$ in polynomial time.*

Proof. We will consider two cases based on the parity of $d/2$ and define appropriate Set Packing instances whose solutions are in a one-to-one correspondence with one-sided d-scattered sets in G. We will then be able to apply the $\sqrt{n}$-approximation for Set Packing of [56]. We will repeat this process for both sides A, B of the bipartition and retain the best solution found. Thus we will be able to approximate $1OPT_d(G)$ within a factor of $\sqrt{n}$ and then rely on Lemma 85 to obtain the claimed bound.

Our Set Packing instances are defined as follows: for $d/2$ even, we make a set c_i for every vertex a_i of A (i.e. from one side) and an element e_j for every vertex b_j of B (i.e. from the other side). For $d/2$ odd, we make a set c_i for every vertex a_i of A (again from one side) and an element e_j for every vertex b_j of B *and* an element r_i for every vertex $a_i \in A$ (i.e. from both sides). Note that $i, j \leq n$. In both cases we include an element corresponding to vertex $x \in G$ in a set corresponding to a vertex $y \in G$, if $d_G(x, y) \leq d/2 - 1$. We then claim that for any given collection C of compatible (i.e. non-overlapping) sets in the above definitions, we can always find a one-sided d-scattered set $S \subseteq A$ in G with $|C| = |S|$ and vice-versa.

First consider the case where $d/2$ is even. Given a one-sided d-scattered set $S \subseteq A$, we let C include all the sets that correspond to some vertex in S and suppose for a contradiction that there exists a pair of sets $c_1, c_2 \in C$ that are incompatible, i.e. that there exists some element e with $e \in c_1$ and $e \in c_2$. Let $a_1, a_2 \in A$ be the vertices corresponding to sets c_1, c_2 and $b \in B$ be the vertex corresponding to element e. Then it must be $d_G(a_1, b) \leq d/2 - 1$ since $e \in c_1$ and $d_G(b, a_2) \leq d/2 - 1$ since $e \in c_2$, that gives $d_G(v_1, v_2) \leq d - 2$, which contradicts S being a d-scattered set. On the other hand, given collection C of compatible sets we let $S \subseteq A$ include all the vertices corresponding to some set in C and suppose there exists a pair of vertices $a_1, a_2 \in S$ for which it is $d_G(a_1, a_2) < d$. Since d is even and $a_1, a_2 \in A$, if $d_G(a_1, a_2) < d$ it must be $d_G(a_1, a_2) \leq d - 2$, as any shortest path between two vertices on the same side of a bipartite graph must be of even length. Thus there must exist at least one vertex $b \in B$ on a shortest path between a_1, a_2 in G for which it is $d_G(a_1, b) \leq d/2 - 1$ and $d_G(b, a_2) \leq d/2 - 1$. This means that the element e corresponding to vertex $b \in B$ must be included in both sets c_1, c_2 corresponding to vertices $a_1, a_2 \in A$, which contradicts the compatibility of sets in C.

We next consider the case where $d/2$ is odd. Given a one-sided d-scattered set $S \subseteq A$, we again let C include all sets that correspond to some vertex in S. If there exists a pair of sets $c_1, c_s \in C$ that contain the same element e corresponding to some vertex $b \in B$ or some element r that corresponds to a vertex $a \in A$, then by the same argument as in the even case we know that there must exist paths of length $\leq d/2-1$ from both vertices $a_1, a_2 \in A$ (corresponding to $c_1, c_2 \in C$) to vertex $b \in B$ or $a \in A$ and thus it must be $d_G(a_1, a_2) < d$. On the other hand, given a collection C of compatible sets we again let $S \subseteq A$ include all the vertices corresponding to sets in C. Supposing there exists a pair $a_1, a_2 \in S$ for which it is $d_G(a_1, a_2) < d$, then again as d is even it must be $d_G(a_1, a_2) \leq d-2$. This means there must be a vertex $a \in A$ on a shortest path between a_1 and a_2 for which $d_G(a_1, a) \leq d/2-1$

and $d_G(a, a_2) \leq d/2 - 1$, which means the corresponding sets $c_1, c_2 \in C$ must both contain element r that corresponds to this vertex $a \in A$, giving a contradiction.

Our algorithm then is as follows. For a given bipartite graph $G = (A \cup B, E)$, we define an instance of Set Packing as described above (depending on the parity of $d/2$) and apply the $\sqrt{n}$-approximation of [56]. Observe that $|A|, |B| \leq n$. We then exchange the sets A, B in the definitions of our instances and repeat the same process. This will return a solution S of size $|S| \geq \frac{1OPT_d(G)}{\sqrt{n}}$, which by Lemma 85 is $\geq \frac{OPT_d(G)}{2\sqrt{n}}$. □

Treewidth of power graphs

We close this chapter with a note on the treewidth of power graphs. Similar ideas as those used in the above results also point to the following upper bound on the increase in treewidth taking place when computing the power of a graph of bounded degree:

Theorem 87. *For any graph G of treewidth tw and maximum degree bounded by Δ, the treewidth tw' of the d-th power G^d is at most $tw' \leq tw \cdot \Delta \sum_{i=0}^{d/2-1}(\Delta - 1)^i = O(tw \cdot \Delta^{d/2})$.*

Proof. Given a tree decomposition T of $G = (V, E)$ of width tw, we make a tree decomposition T' of $G^d = (V, E^d)$ by replacing the appearance of each vertex v in each bag of T with v and the set of vertices at distance at most $d/2$ from v in G, i.e. with $N_G^{d/2}(v) \cup \{v\}$. It is $|N_G^{d/2}(v)| \leq \sum_{i=0}^{d/2-1}(\Delta(\Delta - 1)^i)$, from which we get the upper bound. This is a valid tree decomposition for G^d as: (a) all vertices appear in some bag of T' as they appeared in T, (b) for every edge (u, v) in G^d, either $(u, v) \in E$ and there is a bag in T containing both u, v and thus there is one also in T', or (u, v) was added to E^d due to the distance between u, v being $\leq d$ in G. In this case there must be at least one vertex w at distance $\leq d/2$ from both u and v in G, meaning there will be a bag in T' containing all three vertices u, v, w that was constructed from a bag of T that contains w.

Finally, (c) for every vertex v appearing in two bags X', Y' of T', vertex v also appears on every bag on the path from X' to Y' in T': consider (for a contradiction) the existence of a bag Z' on the path from X' to Y' in T' that does not contain v, and let X, Y, Z be the corresponding bags in T. Since X', Y' contain v, then both X and Y contain some vertex u at distance at most $d/2$ from v in G, or v itself, i.e. $u \in N_G^{d/2}(v) \cup \{v\}$. If both X and Y contain v, then as T is a valid tree decomposition, so does Z and therefore also Z'. Thus we may assume that at least one of X, Y do not contain v, as well as $v \notin Z$. As Z is a separator, then v must appear only on one side of Z in T. We assume (without loss of generality) that v only appears on the X-side of T (from Z) and v is not contained in Y. Thus Y must contain some vertex u at distance $\leq d/2$ from v in G. As Z is a separator, the path from v to u must contain at least one vertex $w \in Z$, at distance $< d/2$ from v. Thus Z' must also contain v, as it includes all vertices at distance $\leq d/2$ from w. □

As for graphs of maximum degree bounded by Δ, we have $\text{cw} \leq O(\Delta \cdot \text{tw})$ (see [32]) and $\text{tw} \leq O(\Delta \cdot \text{cw})$ (directly derived from the well-known result of Gurski and Wanke [54]), we also obtain the following corollary.

Corollary 88. *For any graph G of clique-width cw and maximum degree bounded by Δ, the clique-width cw' of the d-th power G^d is at most $cw' \leq O(cw \cdot \Delta^{d/2+2})$.*

6 Conclusion

Summary

In this thesis we have considered generalizations of the well-known graph-theoretical concepts of independence and domination as they apply to larger distances than simply direct adjacency via edges, by augmenting their definitions with inclusion of a distance parameter. This leads to the computational problems (k, r)-Center and d-Scattered Set that generalize Dominating Set and Independent Set, all of which have been extensively studied before.

We first examined the problems from a parameterized point of view, focusing on structural parameterizations by the most commonly used graph measures treewidth tw, clique-width cw, tree-depth td, vertex cover vc and feedback vertex set fvs, as well as the standard parameterization by the size of the optimal solution. In particular, for (k, r)-Center we showed the following (in Chapter 3):

- A dynamic programming algorithm of running-time $O^*((3r + 1)^{\mathrm{cw}})$ and a matching lower bound based on the SETH, that for $r = 1$ closed a complexity gap for Dominating Set parameterized by cw.

- W[1]-hardness and ETH-based lower bounds of $n^{o(\mathrm{vc}+k)}$ and $n^{o(\mathrm{fvs}+k)}$ for edge-weighted and unweighted graphs, respectively, while these are complemented by an $O^*(5^{\mathrm{vc}})$-time FPT algorithm for the unweighted case.

- An algorithm solving the problem in time $O^*(2^{O(\mathrm{td})^2})$, that assuming the ETH would be optimal as well.

- Algorithms computing for any $\epsilon > 0$ a $(k, (1+\epsilon)r)$-center in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, or $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$, if a (k, r)-center exists in the graph, assuming a tree decomposition of width tw is provided along with the input.

We then turned our attention to d-Scattered Set and applying similar methods showed the following (in Chapter 4):

- A dynamic programming algorithm of running-time $O^*(d^{\mathrm{tw}})$ and a matching lower bound based on the SETH, that generalize known results for Independent Set.

- W[1]-hardness for parameterization by vc + k for edge-weighted graphs, as well as by fvs + k for unweighted graphs, while these are complemented by an FPT-time algorithm for vc and the unweighted case.
- An algorithm solving the problem for unweighted graphs in time $O^*(2^{O(\mathrm{td})^2})$ and a matching ETH-based lower bound.
- An algorithm computing for any $\epsilon > 0$ a $d/(1+\epsilon)$-scattered set in time $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, if a d-scattered set exists in the graph, assuming a tree decomposition of width tw is provided along with the input.

We again note the similarity of our approach here with that of the preceding analysis of the structurally parameterized landscape for (k, r)-Center, as our work on this problem is a continuation of the above. As already discussed, this is partly due to the inherent similarities of both problems when considering distance-based generalizations, as well as the lack of any previous work from this standpoint on either problem. An intuitive observation that can be made, however, is that the distance parameters for our two problems affect our solutions in a complementary manner, in much the same way as vertex adjacency antithetically affects the concepts of domination and independence, yet the apparent 'duality' between them makes handling their generalizations over larger distances susceptible to comparable techniques.

Finally, we advanced our investigation of the d-Scattered Set problem by answering some remaining questions on its (super-)polynomial (in-)approximability. In particular, we showed the following (in Chapter 5):

- An exact exponential-time algorithm of complexity $O^*((ed)^{\frac{2n}{d}})$.
- A lower bound on the complexity of any ρ-approximation algorithm of (roughly) $2^{\frac{n^{1-\epsilon}}{\rho d}}$ for even d and $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ for odd d, under the randomized ETH.
- ρ-approximation algorithms of running-times $O^*((e\rho d)^{\frac{2n}{\rho d}})$ for even d and $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ for odd d that (almost) match the above lower bounds, thus giving a clear picture of the trade-off curve between approximation and running-time.
- A lower bound of $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ on the approximation ratio of any polynomial-time algorithm for graphs of maximum degree Δ and an improved upper bound of $O(\Delta^{\lfloor d/2 \rfloor})$ on the approximation ratio of any greedy scheme for this problem, that matches our lower bound.
- A polynomial-time approximation algorithm of ratio $2\sqrt{n}$ for bipartite graphs and even values of d, that complements known results by considering the only remaining open case.

We have already observed in both cases that the distance parameter's influence on the complexity of the problem is not negligible, as both (k, r)-Center and d-Scattered Set become harder than Dominating Set and Independent Set when r or d is unbounded: our W-hardness results for parameterization by k and vc (weighted), fvs (unweighted) dictate that if the distance parameter takes large values, even the graph's (considerably) restricted structure will not be of much help in significantly improving upon the already attainable XP running-tmes.

Considering both these problems have already been proven intractable by almost all other computational alternatives to exact computation (see Section 2.2), while also taking into account specifically for d-Scattered Set, its $\Delta^{d/2}$-inapproximability in polynomial time where d appears on the exponent of the ratio, as well as the fact that for large enough d even the PTAS for planar graphs from [43] is not applicable (see below), it seems that in both cases a good way to approach these problems when the distance parameters are large is to apply either our FPT algorithms for the unweighted case parameterized by vc/td, or our tw/cw-based FPT approximation schemes whose running-times only depend on the structure of the input graph. Note that both require significant restrictions on the input, while the former is incompatible with the important weighted case and the latter can indeed correctly identify the size of the optimal solution, albeit with admissible loss of precision in terms of satisfaction of distance requirements.

Open problems

Given the intractability and inapproximability of the standard parameterizations of both problems, as well as our SETH-based lower bounds on their structurally parameterized complexity, we are subsequently interested in the best achievable ratios by *structurally parameterized approximation algorithms* of running-times that clearly improve upon the lower bounds of the exact cases given here. As a concrete example, what would be the best achievable ratio ρ by a 2^{tw}-time ρ-approximation algorithm for either problem where the value of r/d is unbounded?

Apart from this direction and considering also previous related work (notably [22]), the above results can be seen to complete the picture on the (k, r)-Center problem's structurally parameterized complexity. Nevertheless, some remaining open questions concern the sharpening of our ETH-based lower bounds using as a starting point the more precise SETH, as well as the complexity status of the problem with respect to other structural parameters, such as rankwidth, modular-width or neighborhood diversity.

Remaining open questions on the structurally parameterized complexity of the d-Scattered Set problem may concern the identification of similarly tight upper and lower (SETH-based) bounds for clique-width (that is FPT for $d = 2$), as well as the sharpening of our ETH-based lower bounds for vc and fvs, that are not believed to be tight due to the quadratic blow-up in parameter size in our reductions.

On our (super-)polynomial work on d-Scattered Set, apart from the possibility of "de-randomization" of the results above that use the randomized construction of [26] as a starting point, some unanswered questions involve:

- the sharpening of our super-polynomial upper bounds to exactly match the lower bounds, i.e. ρ-approximations in time $O^*(2^{\frac{2n}{\rho d}})$ for even d and $O^*(2^{\frac{2n}{\rho(d+\rho)}})$ for odd d, noting that even algorithms of running-times keeping the same exponent but where the base does not depend on d would hint at the problem in fact becoming easier to approximate for large enough values of d;

- the complexity of the problem on chordal bipartite graphs, also mentioned as an open problem by [42];

- the functionality of the PTAS for planar graphs by the same authors, that only works for fixed values of d, as it extends the well-known approach of [6] for obtaining such algorithms for several problems, including Independent Set.

Because the technique of [6] involves breaking down the graph into (roughly) d-outerplanar subgraphs and then exactly solving the problem in each of these using dynamic programming over their tree decompositions, for values of d that are not constant (say $d \geq \sqrt{n}$)) this is not achievable in polynomial-time due to the exponent of the treewidth algorithms depending on d. It would be interesting to see an extension of this (or some other) approach for the case of unbounded d, or, conversely, a hardness reduction proving it is unlikely. The difficult part here would have to involve a construction that is very efficient in terms of *crossing* gadgets in order to maintain planarity, or, from the other side, a way to optimally solve the problem in carefully constructed subgraphs without the exponential requirement on d. Note this is intuitively related to whether the problem becomes easier to approximate with large d, a question also raised by our first point above.

7

Résumé des chapitres en français

Introduction

Traçabilité et Hypothèses de complexité: L'objectif de la théorie de la complexité computationnelle est la catégorisation des problèmes mathématiques en classes selon les temps d'exécution les plus défavorables des algorithmes qui les résolvent. Dans le cadre classique, les problèmes sont considérés *tractables*, c'est-à-dire résolvables en temps polynomial, s'il existe un algorithme dont le temps d'exécution peut être exprimé sous la forme d'une fonction polynomiale sur la taille n de l'entrée.

D'autre part, les problèmes *insolubles* (généralement NP-difficile) sont ceux pour lesquels un algorithme en temps polynomial est considéré comme peu probable, sur la base de la conjecture (largement répandue) de P≠NP: si le problème 3-SAT n'admet pas un algorithme de temps polynomial déterministe, alors une *réduction* de 3-SAT à un autre problème,c'est-à-dire une transformation des instances d'un problème en celles de l'autre démontrant leur équivalence en termes de complexité de calcul, impliquerait que ce dernier problème n'admet pas également un algorithme en temps polynomial.

L'approfondissement des considérations ci-dessus conduit à la formulation d'une autre conjecture (également largement répandue), *l'hypothèse de temps exponentiel* (ETH): l'hypothèse conjectures il n'y a pas d'algorithme *subexponentiel* pour 3-SAT, c'est à dire pas d'algorithme de temps de fonctionnement $2^{o(n)} \cdot n^{O(1)}$. Si cette hypothèse est vraie, alors P n'est pas égal à NP et tout algorithme pour 3-SAT nécessitera au moins un temps exponentiel, dans le pire des cas.

Une version un peu plus exigeante (et pas si répandue), la *robuste hypothèse de temps exponentiel* (SETH) a également été formulée et affirme que (général) SAT n'admet pas un algorithme de temps de fonctionnement $(2-\epsilon)^n \cdot n^{O(1)}$ pour toute constante $\epsilon > 0$.

Limites supérieures/inférieures: Comme pour l'hypothèse que P≠NP, l'importance principale de l'ETH et de la SETH ne réside cependant pas dans le fait que ceux-ci puissent être réellement vrais ou non : de la même manière que pour la classification des problèmes en temps polynomial résoluble ou NP-difficile, nous pouvons utiliser l'ETH et la SETH comme points de départ pour montrer, par des réductions de difficulté, l'inexistence de tout algorithme d'un certain temps de fonctionnement *spécifique* en dessous d'un certain seuil et ainsi obtenir des résultats qui excluent l'existence de tels algorithmes pour un problème donné (*une limite inférieure*).

En combinant des résultats de ce type avec des algorithmes dont les temps de fonctionnement les plus défavorables (*limite supérieure*) correspondent exactement à ces limites inférieures, nous pouvons identifier précisément la complexité d'un problème donné et justifier la *optimalité* de notre approche. Naturellement, toute déclaration de ce type que nous ferons sera soumise aux hypothèses ci-dessus, ce qui signifie que nos résultats impliqueront que les algorithmes proposés sont optimaux, à moins que des progrès significatifs ne soient réalisés dans notre compréhension des principes fondamentaux du calcul.

De cette façon, nous pouvons classer davantage les problèmes en fonction de leurs exigences de calcul, en particulier dans le cas du SETH qui offre une source plus précise au prix d'une hypothèse plus ambitieuse (et donc plus susceptible d'être fausse). Ce raffinement nous permet de mieux comprendre les options disponibles pour résoudre un problème de calcul dont la complexité est prouvée et peut potentiellement conduire à des améliorations pratiques des paramètres appliqués, mais c'est la possibilité de caractériser précisément les caractéristiques de complexité sous-jacentes des problèmes mathématiques qui nous intéressera le plus ici.

Ayant montré à la fois une limite supérieure et une limite inférieure de fonctions de complexité correspondantes pour un problème donné (c'est-à-dire des limites qui sont "strictes"), nous pouvons généralement identifier une uniformité dans la structure mathématique des deux preuves qui n'est pas arbitraire, comme dans l'optimalité d'une réduction qui produira des instances (presque) explicitement construites pour entraver les efforts de l'algorithme dont le temps de fonctionnement correspond exactement à la limite inférieure de la réduction (et vice-versa). De tels résultats peuvent être considérés comme impliquant qu'un aspect de l'essence du problème a été indéniablement identifié, puisque leur validité ne dépend pas des méthodes particulières employées par leur concepteur.

Paramétrage et approximation Le fait de pouvoir caractériser l'intractibilité d'un problème selon un mode de calcul particulier ne signifie pas que nous avons épuisé toutes les possibilités pour le résoudre. Le fait de considérer un problème comme insoluble si le temps d'exécution requis de tout algorithme pour sa solution exacte est au moins exponentiel dans la taille de l'entrée peut donc conduire à d'autres directions pour faire progresser notre compréhension des mécanismes compliqués qui régissent les problèmes combinatoires complexes : *paramétrage* et *approximation*.

D'une part, et à la recherche d'une caractérisation plus fine de la quantité de calcul nécessaire pour la résolution exacte et optimale d'un problème de calcul, nous pourrions permettre aux fonctions de complexité de croître effectivement de manière exponentielle, mais pas sur la taille n de l'entrée (qui doit naturellement être considérée comme trop importante et peu pratique). Par le biais du paramétrage, nous étudions la complexité des problèmes en fonction d'autres paramètres qui spécifient leurs propriétés que la simple taille de l'entrée, paramètres dont la taille (limitée) n'empêcherait pas le calcul pratique d'un nombre exponentiel.

D'autre part, en assouplissant l'exigence selon laquelle les solutions renvoyées par nos algorithmes sont nécessairement les meilleures possibles (étant d'une qualité mesurable pour les problèmes d'"optimisation") et en se concentrant sur le maintien du polynôme des temps de fonctionnement sur la taille de l'entrée, nous entrons dans le domaine de l'approximation. Ici, nos solutions doivent être accompagnées de garanties mathématiques de rester au-dessus de certains seuils de qualité (un *ratio d'approximation* pire).

Dans la complexité paramétrée, de manière similaire à la classification des problèmes

comme NP-difficile ou polynomial-temps résoluble, un problème mathématique qui est résolu par un algorithme dont la complexité peut être exprimée en fonction de la forme $f(k) \cdot n^{O(1)}$, où k est le paramètre choisi et f est une fonction calculable quelconque, appartient à la classe des problèmes *tractables à paramètre fixe* (FPT). Selon le problème, les fonctions f peuvent prendre de nombreuses formes, étant exponentielles dans la majorité des cas.

Cela implique que si la taille du paramètre considéré n'est pas trop importante, pour un cas donné de problème à résoudre, alors un algorithme FPT qui le résout pourrait être considéré comme utilisable (peut-être même pratique), tout en apportant des améliorations importantes au paysage de la complexité en général.

Les paramètres communs comprennent la taille d'une solution optimale (la paramétrisation *standard*), ainsi qu'une variété de mesures *structurales* qui caractérisent la structure inhérente de l'instance d'entrée. Ici, un problème est considéré comme insoluble s'il peut être démontré (via les *réductions paramétrées*, en maintenant également une relation étroite entre les paramètres) qu'il est aussi difficile à résoudre que tout problème complet pour un niveau de la *hiérarchie W* des classes de complexité (considéré comme un analogue de la difficulté du NP), c'est-à-dire s'il est peu probable qu'il soit FPT.

La théorie des algorithmes d'approximation s'intéresse à l'aspect complémentaire de l'intractibilité : l'identification des meilleures limites possibles du pire cas sur la qualité d'une solution retournée qui peut être obtenue si le temps de fonctionnement est limité aux polynômes en n. Ces limites sont généralement exprimées comme le ratio entre la qualité la plus mauvaise d'une solution retournée et celle d'une solution optimale pour le même cas. Du côté de la "difficulté", il est possible de montrer (via *réductions préservant l'approximation*) qu'il n'existe pas d'algorithme de temps polynomial atteignant un certain ratio pour un problème donné, sous des hypothèses de complexité standard, établissant ainsi sa *inapproximabilité*.

Les ratios courants dans les résultats de ce type comprennent l'inaptitude à se rapprocher de constantes spécifiques, de tout ratio constant possible, ainsi que d'un ratio qui est une fonction de n. Le temps d'exécution autorisé pour un algorithme d'approximation est généralement polynomial en n, mais il est possible d'autoriser d'autres fonctions (telles que les temps d'exécution FPT) afin de proposer des alternatives au calcul exact, si un problème reste insoluble au-delà de la limite du temps polynomial.

Pour en revenir à l'ETH et la SETH, on peut observer qu'en conjonction avec l'analyse de complexité affinée effectuée par les études de paramétrage, il est possible d'obtenir des limites inférieures améliorées de précision accrue sur le temps d'exécution requis de tout algorithme pour un problème donné. Les deux hypothèses peuvent être considérées comme des hypothèses sur la complexité de q-SAT paramétrée par le nombre de variables n et une réduction paramétrée à un autre problème dans ce cas (c'est-à-dire où la taille du paramètre est limitée par une fonction appropriée de n) donnerait des résultats sur l'inexistence d'un algorithme sous-exponentiel (dans la taille du paramètre) pour le problème en question.

Ainsi, les problèmes paramétrés peuvent être davantage catégorisés en termes de fonctions exactes qui déterminent leur complexité par rapport à la variété de paramètres possibles qui participent à leur intractibilité, ce qui conduit à une compréhension bien plus grande du domaine de la complexité computationnelle.

Problèmes de couverture et d'emballage: Dans cette thèse, nous nous concentrons sur les problèmes bien connus de la théorie des graphes (k, r)-Center et d-Scattered Set qui généralisent les concepts de dominance et indépendance sur de plus grandes distances à l'intérieur du graphe. Dans le problème Dominating Set, nous recherchons le sous-ensemble *le plus petit* de sommets, de sorte que chaque sommet *autre* soit relié à au moins un sommet du sous-ensemble. En revanche, dans les IND, nous avons besoin du sous-ensemble *plus grand* de telle sorte qu'aucune paire de sommets *dans le sous-ensemble* ne soit reliée par un bord.

Intuitivement, un ensemble dominant doit *couvrir* le reste du graphe sur la base de la contiguïté combinée de ses sommets à ceux du complément, tandis que dans un ensemble indépendant, nous devons pouvoir *emballer* autant de sommets non reliés par paires que possible. Les deux problèmes sont fermement insolubles : ils sont NP-difficile, leurs paramétrages standard W-difficile et généralement inapprochables en temps polynomial-ainsi qu'en temps FPT. D'un point de vue positif, les deux problèmes se révèlent être FPT lorsqu'ils sont paramétrés par les paramètres structurels les plus couramment utilisés. Cela signifie que lorsque le graphe d'entrée est de structure restreinte, les deux problèmes peuvent être efficacement, ainsi que avec précision résolus.

Les généralisations de ces notions bien étudiées que nous examinons ici sont basées sur l'extension du *paramètre de distance* central dans leurs définitions à des valeurs non limitées. Dans (k, r)-Center nous devons indiquer le plus petit ensemble qui couvre le graphe *à distance* r et dans d-Scattered Set nous devons regrouper autant de sommets que possible *à distance* d les uns des autres.

Cela signifie que notre perspective ici doit s'élargir pour considérer l'influence des sommets participant à la solution sur de plus grandes zones dans le graphe, car l'importance de la contiguïté réside maintenant dans les *chemins* au lieu des arêtes. En guise de remarque préliminaire, il s'avère que l'accessibilité entre les sommets est une propriété trop sensible aux petites variations de leur emplacement exact, ce qui fait que l'existence de collections de chemins de longueur non négligeable dépend de manière cruciale de la forme exacte des structures *locales* et que, par conséquent, le comportement des deux problèmes diverge (de manière significative) de leur cas de base lorsque les r, d sont importants. Cet effet se reflète dans nos résultats, puisque les deux problèmes deviennent insolubles même pour des graphes de structure significativement restreinte, si la valeur du paramètre de distance n'est pas limitée dans chaque cas. Ainsi, les algorithmes susmentionnés ne sont efficaces que pour les petites valeurs fixes (par exemple, $r = 1, d = 2$), ce qui motive notre analyse.

Notre champ d'action: Nous examinons les problèmes (k, r)-Center et d-Scattered Set, en accordant une attention particulière à la manière dont leur complexité est affectée par les paramètres de distance et aux options disponibles pour leur calcul exact et/ou efficace. Comme nos problèmes sont en fait des généralisations de Dominating Set et Independent Set, on peut considérer que nos résultats correspondent (et parfois même améliorent) l'état de l'art pour ces problèmes.

Dans la première partie de la thèse, nous maintenons un point de vue paramétré : nous examinons le paramétrage standard, ainsi que les mesures de graphes les plus couramment utilisées : treewidth tw, clique-width cw, tree-depth td, vertex cover vc et feedback vertex set fvs. Nous proposons des résultats de difficulté qui montrent qu'il n'y a pas d'algorithme de temps d'exécution en dessous de certaines limites (sous réserve de l'ETH, SETH), produisons des algorithmes essentiellement optimaux de complexité qui

correspondent à ces limites inférieures et tentons en outre d'offrir une alternative au calcul exact dans un temps d'exécution considérablement réduit par approximation.

En particulier, pour (k, r)-Center nous montrons ce qui suit :

- Un algorithme de programmation dynamique du temps d'exécution $O^*((3r+1)^{\mathrm{cw}})$, en supposant qu'une expression clique-width de largeur cw est fournie avec l'entrée, et une borne inférieure correspondante basée sur SETH qui comble un écart de complexité pour Dominating Set paramétré par cw (pour $r = 1$).
- W[1]-difficulté et limites inférieures basées sur l'ETH de $n^{o(\mathrm{vc}+k)}$ pour les graphes à bords pondérés et $n^{o(\mathrm{fvs}+k)}$ pour les graphes non pondérés. Cela montre l'importance de délimiter la valeur de r. Toujours pour le cas non pondéré, nous donnons un algorithme FPT $O^*(5^{\mathrm{vc}})$-temps basé sur la résolution des sous-instances Set Cover appropriées.
- Une limite inférieure stricte basée sur l'ETH de $O^*(2^{O(\mathrm{td})^2})$ pour le paramétrage par td.
- Algorithmes calculant pour tout $\epsilon > 0$, un $(k, (1+\epsilon)r)$-centre dans le temps $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, ou $O^*((\mathrm{cw}/\epsilon)^{O(\mathrm{cw})})$, si un (k, r)-centre existe dans le graphe, en supposant qu'une décomposition en arbre de largeur tw est fournie avec l'entrée.

Puis pour d-Scattered Set et en appliquant des méthodes similaires nous montrons :

- Un algorithme de programmation dynamique du temps d'exécution $O^*(d^{\mathrm{tw}})$ et une limite inférieure correspondante basée sur la SETH, qui généralisent les résultats connus pour Independent Set.
- W[1]-difficulté pour le paramétrage par vc$+k$ pour les graphes à bords pondérés, ainsi que par fvs$+k$ pour les graphes non pondérés, ce qui montre à nouveau l'importance de délimiter d. Ils sont complétés par des algorithmes FPT-temps pour le cas non pondéré qui utilisent des idées liées à Set Packing, de complexité $O^*(3^{\mathrm{vc}})$ pour les d pairs et $O^*(4^{\mathrm{vc}})$ pour les d impairs.
- Une limite inférieure stricte basée sur l'ETH de $O^*(2^{O(\mathrm{td})^2})$ pour le paramétrage par td, comme ci-dessus.
- Un algorithme calculant, pour tout $\epsilon > 0$, un ensemble $d/(1+\epsilon)$-diffusé dans le temps $O^*((\mathrm{tw}/\epsilon)^{O(\mathrm{tw})})$, si un ensemble d-diffusé existe dans le graphe, en supposant qu'une décomposition en arbre de largeur tw est fournie dans l'entrée.

Nous notons que ces résultats sont comparables à ceux de (k, r)-Center, puisque notre travail sur d-Scattered Set peut être considéré comme une continuation de ce qui précède. Comme nous le voyons, les deux problèmes sont affectés de manière similaire par des généralisations basées sur la distance et sont donc sensibles à des techniques similaires.

Dans la deuxième partie de la thèse, nous nous concentrons sur d-Scattered Set et en particulier sur son (in)approximabilité (super-)polynomiale : nous nous intéressons à la relation exacte entre un ratio d'approximation réalisable ρ, le paramètre de distance d, et le temps de fonctionnement de tout algorithme d'approximation ρ exprimé en fonction

de ce qui précède et la taille de l'entrée n. Ensuite, nous considérons des temps de fonctionnement strictement polynomiaux et des graphes de degré maximal borné ainsi que des graphes bipartites. Plus précisément, nous montrons :

- Un algorithme exact en temps exponentiel de complexité $O^*((ed)^{\frac{2n}{d}})$, basé sur une limite supérieure de la taille de toute solution.
- Une limite inférieure à la complexité de tout algorithme de ρ-approximation de (environ) $2^{\frac{n^{1-\epsilon}}{\rho d}}$ pour les d pairs et de $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ pour les d impairs, sous l'ETH randomisé.
- Les algorithmes de ρ-approximation des temps de fonctionnement ρ $O^*((e\rho d)^{\frac{2n}{\rho d}})$ pour les d pairs et $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ pour les d impairs qui correspondent (presque) aux limites inférieures ci-dessus.
- Une limite inférieure de $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ sur le ratio d'approximation de tout algorithme de temps polynomial pour les graphes de degré maximal Δ, étant la première limite inférieure de ce type, ainsi qu'une limite supérieure améliorée de $O(\Delta^{\lfloor d/2 \rfloor})$.
- Une approximation polynomiale en temps de $2\sqrt{n}$ pour les graphes bipartites et pour les valeurs paires de d, qui complète les résultats connus en considérant le seul cas ouvert restant.

Le problème (k, r)-Center

Au chapitre 3, nous étudions le problème (k, r)-Center. Il s'agit d'un problème d'optimisation extrêmement bien étudié, avec de nombreuses applications. Il a une longue histoire, surtout du point de vue de l'approximation des algorithmes, dont l'objectif est généralement de minimiser les r pour un k donné [1, 41, 45, 59, 69, 70, 72, 85, 95]. L'objectif inverse (minimiser les k pour un r donné) a également été bien étudié, le problème étant généralement appelé r-Dominating Set dans le cas présent [24, 30, 50, 77, 91].

Parce que (k, r)-Center généralise Dominating Set (ce qui correspond au cas $r = 1$), le problème peut déjà être considéré comme difficile, voire approximatif (sous hypothèses de complexité standard). Comme cette difficulté persiste lorsque l'on considère la paramétrisation standard du problème avec des approximations paramétrées non triviales également exclues, nous sommes fortement motivés à étudier la complexité du problème lorsque le graphe d'entrée a une structure restreinte.

Nos résultats: Notre objectif est d'effectuer une analyse complète de la complexité de (k, r)-Center qui prend en compte la structure de cette entrée en utilisant le cadre de complexité paramétrée. En particulier, nous fournissons les résultats des limites supérieure et inférieure *à grain fin* de la complexité de (k, r)-Center par rapport aux des paramètres largement étudiés qui mesurent la structure d'un graphe : treewidth **tw**, clique-width **cw**, tree-depth **td**, vertex cover **vc**, et feedback vertex set **fvs**.

En plus de la valeur intrinsèque de la détermination de la complexité précise du (k, r)-Center, cette approche est en outre motivée par le fait que les algorithmes FPT pour ce problème ont souvent ont été utilisés comme éléments de base pour des algorithmes d'approximation plus élaborés [37, 41]. En effet, (certaines de) ces questions ont déjà

été pris en compte, mais nous fournissons un certain nombre de nouveaux résultats qui s'appuient sur et améliorer l'état actuel des connaissances.

En cours de route, nous comblons également un vide sur la complexité du problème emblématique Dominating Set paramétré par clique-width. Plus précisément, nous prouvons ce qui suit :

- (k, r)-Center peut être résolu (sur des graphes non pondérés) dans le temps $O^*((3r+1)^{\mathrm{cw}})$ (si une expression de clique-width est fournie avec l'entrée), mais il ne peut pas être résolu dans le temps $O^*((3r+1-\epsilon)^{\mathrm{cw}})$ pour tout $r \geq 1$ fixe, sauf si la SETH échoue.

  Le résultat algorithmique s'appuie sur des techniques standard (programmation dynamique sur des expressions de clique-width, convolution rapide de sous-ensembles), ainsi que sur plusieurs observations spécifiques au problème qui sont nécessaires pour obtenir la taille de table souhaitée. La limite inférieure de SETH résulte d'une réduction directe de SAT.

  Une conséquence notable de notre résultat à la limite inférieure est que, dans le cas de Dominating Set, il comble l'écart entre la complexité des meilleurs algorithmes connus ($O^*(4^{\mathrm{cw}})$ [15]) et le meilleur précédemment limite inférieure connue ($O^*((3-\epsilon)^{\mathrm{cw}})$ [76]).

- (k, r)-Center ne peut être résolu à temps $n^{o(\mathrm{vc}+k)}$ sur les graphes pondérés par les bords, ou temps $n^{o(\mathrm{fvs}+k)}$ sur des graphes non pondérés, sauf si l'ETH est fausse.

  On savait déjà qu'un algorithme FPT paramétré juste par tw (pour les r non limités) est peu probable [22].

  Ces résultats montrent qu'il en va de même pour les deux autres paramètres restrictifs fvs et vc, même si k est également ajouté comme paramètre. Ils sont (asymptotiquement) stricts, car il est facile d'obtenir algorithmes $O^*(n^{\mathrm{fvs}})$, $O^*(n^{\mathrm{vc}})$, et $O^*(n^k)$.

  Nous remarquons que (k, r)-Center est un exemple rare de problème qui s'avère être difficilement paramétrable par vc. Nous complétons ces limites inférieures par un algorithme FPT pour le cas non pondéré, en cours d'exécution dans le temps $O^*(5^{\mathrm{vc}})$.

- (k, r)-Center peut être résolu dans le temps $O^*(2^{O(\mathrm{td}^2)})$ pour les graphes non pondérés, mais s'il peut être résolu dans le temps $O^*(2^{o(\mathrm{td}^2)})$, alors l'ETH est faux.

  Ici, la limite supérieure découle des connexions connues entre la tree-depth d'un graphe et son diamètre, tandis que la limite inférieure découle d'une réduction de 3-SAT. Nous remarquons qu'il s'agit d'un exemple quelque peu inhabituel de problème paramétré dont la dépendance aux paramètres s'avère être exponentielle dans le *carré* du paramètre.

Les résultats ci-dessus, ainsi que les travaux récents de la [22] qui ont montré des limites étroites de $O^*((2r+1)^{\mathrm{tw}})$ concernant la complexité du problème paramétrée par tw, donnent une image complète, et souvent à grain fin, image sur (k, r)-Center pour les paramètres les plus importants du graphe. L'un des les conclusions que l'on peut tirer sont que, en raison de la difficulté pour vc (dans le cas pondéré) et fvs, il y a peu les cas où l'on peut espérer obtenir un algorithme FPT sans en limiter la valeur de r. En

d'autres termes, à mesure que r augmente la complexité de la résolution exacte du problème dégénère rapidement et s'éloigne de la cas de Dominating Set, qui est FPT pour tous les paramètres considérés.

Une autre contribution de ce chapitre est de compléter cette opinion négative en soulignant qu'elle ne s'applique que si l'on insiste pour résoudre le problème *exactement*.

Si nous autorisons les algorithmes qui renvoient un $(1+\epsilon)$-approximation du r optimal, pour un $\epsilon > 0$ arbitrairement petit et tout en respectant la valeur donnée de k, on obtient le suivant :

- Il existe des algorithmes qui, pour tout $\epsilon > 0$, lorsque l'on donne un graphe qui admet un (k, r)-centre, renvoient un $(k, 1+\epsilon)r)$-centre dans le temps $O^*((tw/\epsilon)^{O(\mathrm{tw})})$, ou $O^*((cw/\epsilon)^{O(\mathrm{cw})})$, en supposant qu'une décomposition en arbre ou une expression de clique-width est donnée dans l'entrée.

L'algorithme d'approximation tw est basé sur une technique introduite dans [73], tandis que l'algorithme cw repose sur une nouvelle extension d'une idée de [54], qui peut présenter un intérêt indépendant.

Grâce à ces algorithmes d'approximation, nous arrivons à une meilleure compréhension de la complexité de (k, r)-Center en incluant la question de l'approximation, et obtenons des algorithmes qui continuent à fonctionner efficacement même pour des valeurs importantes de r. La figure 7.1 illustre les relations entre les paramètres et le tableau 7.1 résume nos résultats.

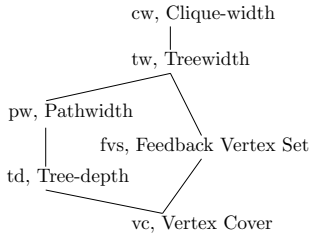


Figure 7.1: Relations des paramètres. Les résultats algorithmiques sont hérités à la baisse, de la difficulté à la hausse.

	cw	tw	fvs	td	vc
FPT exact	24 (w/u)	46 (w/u)		57 (u)	29 (u)
FPT-AS	45 (w/u)	42 (w/u)			
SETH LB	21 (u)				
ETH LB			28 (w/u)	35 (u)	27 (w)
W[1]-hard			28 (w/u)		27 (w)

Table 7.1: Un résumé de nos résultats (nombres de théorème) pour tous les paramètres considérés. Les initiales u/w désignent les variantes non pondérées/pondérées du problème.

Le problème d-Scattered Set

Au chapitre 4, nous étudions le problème d-Scattered Set. Le problème peut déjà être considéré comme difficile, car il généralise Independent Set (pour $d = 2$), alors qu'un autre nom est Distance-d Independent Set [43, 81, 42]. Cette difficulté incite à l'analyse du problème lorsque le graphe d'entrée est de structure restreinte, notre objectif étant de fournir un compte rendu complet de la complexité de d-Scattered Set par le biais de divers résultats de limites supérieure et inférieure.

Notre point de vue est paramétré : nous considérons que le fameux paramètres structurels treewidth **tw**, tree-depth **td**, vertex cover **vc** et feedback vertex set **fvs**, qui expriment principalement les restrictions prévues sur la structure du graphe d'entrée, tandis que nous examinons à la fois les variantes pondérées et non pondérées.

Avant de décrire nos résultats en détail, nous notons qu'ils suivent une démarche similaire à celle du chapitre précédent. En effet, notre analyse des propriétés structurellement paramétrées du problème d-Scattered Set est analogue à celle effectuée pour (k, r)-Center et nos résultats sont en fait comparables. Cela est dû en partie à la proximité des techniques utilisées pour les obtenir, mais peut également être considéré comme une conséquence de la similitude de l'influence des généralisations basées sur la distance sur les fonctions d'indépendance et de domination.

La principale observation que l'on peut faire sur cette similitude est peut-être que, outre sa signification inverse pour chaque problème (c'est-à-dire que les distances doivent être soit $\geq d$ soit $\leq r$), les paramètres de distance mesurent dans chaque cas l'étendue de l'influence qu'un sommet peut exercer sur sa région dans le graphe et impliquent donc une division du paysage en "zones d'influence", qui doivent être efficacement agencées (c'est-à-dire emballées ou couvertes) pour être optimales.

À titre d'exemple, cela se reflète dans nos algorithmes de programmation dynamique (Théorème 24 et Théorème 51, voir également Théorème 46 et [22, 50]) et les représentations d'état qu'ils emploient, avec une correspondance apparente entre l'importance des distances $d/2$ et r [1]. La justification de la nécessité de ces représentations d'état est donnée par les limites inférieures correspondantes (Théorème 21 et Théorème 50), ce qui signifie que notre approche n'est pas déraisonnablement applicable dans chaque cas, unifiant ainsi également la classification de la complexité structurellement paramétrée des deux problèmes.

Nos résultats: Premièrement, dans la sous-section 4.1.1 nous présentons une limite inférieure de $(d-\epsilon)^{\mathrm{tw}} \cdot n^{O(1)}$ sur la complexité de tout algorithme résolvant le problème d-Scattered Set paramétré par treewidth tw, sur la base du SETH. Ce résultat peut être considéré comme un extension de la borne de $(2-\epsilon)^{\mathrm{tw}} \cdot n^{O(1)}$ pour Independent Set ([76]) pour des valeurs plus importantes de d, pour lesquelles la construction doit être beaucoup plus compacte en termes d'informations codées par unité de treewidth.

Dans la sous-section 4.1.2, nous fournissons une algorithme de programmation du temps d'exécution $O^*(d^{\mathrm{tw}})$, correspondant à cette limite inférieure, sur une décomposition d'arbre donné de largeur tw. L'algorithme résout en fait la version de comptage de d-Scattered Set, en utilisant des techniques standard (programmation dynamique sur les décompositions d'arbres), avec une application de la technique de convolution rapide de sous-ensembles de [9] (ou *changements d'état* [15, 94]) pour faire correspondre le temps de fonctionnement à la taille des tables de programmation dynamique.

Ayant ainsi identifié la complexité du problème par rapport à tw, nous nous concentrons ensuite sur les paramètres plus généraux vc et fvs et nous montrons dans la sous-section 4.2.1 que le problème d-Scattered Set pondéré par les bords et paramétré par $\mathrm{vc}+k$ est W[1]-difficile. Si, en revanche, tous les poids de bord sont réglés sur 1, alors d-Scattered Set (la variante non pondérée) paramétrée par $\mathrm{fvs}+k$ est W[1]-difficile. Nos réductions impliquent également des limites inférieures exponentielles basées sur l'ETH sur la racine carrée des paramètres, mais nous ne pensons pas que celles-ci soient strictes (par opposition à une limite inférieure simplement exponentielle), en raison de l'augmentation quadratique de la taille des paramètres de notre construction (puisque notre attention se porte sur les bords).

[1] Cette "dualité" entre les problèmes et les distances r et $d/2$ est également abordée dans [88].

Nous complétons ces résultats par un algorithme exponentiel unique pour la variante non pondérée, du temps de fonctionnement $O^*(3^{\text{vc}})$ pour le cas de d pairs, tandis que pour les d impairs, le temps de fonctionnement est de $O^*(4^{\text{vc}})$. L'algorithme repose sur définir une variante de Set Packing comme un sous-problème que nous résolvons via la programmation dynamique, avec des durées de fonctionnement différentes selon les parité de d, étant donné le nombre de situations possibles pour un sommet avec en ce qui concerne les candidats potentiels à la sélection.

Ensuite, pour la variante non pondérée, nous indiquons également Sous-section 4.2.3 l'existence d'un algorithme paramétré par td de temps de fonctionnement $O^*(2^{O(\text{td}^2)})$, ainsi qu'une limite inférieure correspondante basée sur l'ETH. La limite supérieure découle des connexions connues entre la tree-depth d'un graphe et de son diamètre, tandis que la limite inférieure provient d'une réduction de 3-SAT.

Enfin, nous revenons à tw dans la section 4.3 et nous présentons un FPT-AS du temps de fonctionnement $O^*((\text{tw}/\epsilon)^{O(\text{tw})})$ qui trouve un ensemble $d/(1+\epsilon)$-dispersé de taille k, si un ensemble d-dispersé de même taille existe.

L'algorithme est basé sur un technique d'arrondissement introduite dans [73] et on peut voir surpasse tout algorithme exact pour le problème (pour les gros d, c'est-à-dire $O(\log n)$), même pour le cas non pondéré et les paramètres plus restreints (de manière similaire à la section 3.3). La figure 7.2 montre les relations hiérarchiques entre les paramètres et le tableau 7.2 résume nos résultats.

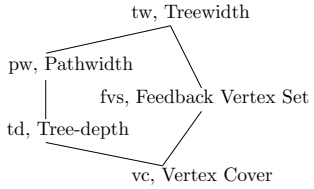


Figure 7.2: Relations entre les paramètres considérés. Les résultats algorithmiques sont hérités à la baisse (de cw à vc), de la difficulté à la hausse.

	tw	fvs	td	vc
FPT exact	51 (w/u)		57 (u)	56 (u)
FPT-AS	63 (w/u)			
SETH LB	50 (w/u)			
ETH LB		55 (w/u)	61 (u)	54 (w)
W[1]-difficile		55 (w/u)		54 (w)

Table 7.2: Un résumé de nos résultats pour tous les paramètres considérés. Les initiales u/w désignent les variantes non pondérées/pondérées.

Approximation du problème d-Scattered Set

Dans le chapitre 4, nous réexaminons le problème d-Scattered Set, mais nous nous concentrons ici sur les aspects du problème qui concernent davantage son approximation que la sensibilité de sa complexité aux restrictions de la structure de l'entrée. Nous considérons les temps d'exécution exprimés sous forme de fonctions de n, la taille de l'instance, ainsi que d'autres paramètres pertinents, tandis que nous nous intéressons aux temps d'exécution polynomiaux, ainsi que super-polynomiaux (c'est-à-dire exponentiels). En outre, nous nous concentrons sur la variante non pondérée du problème, où la distance est mesurée uniquement par le nombre d'arêtes.

Après avoir examiné la complexité du problème par rapport aux paramètres structurels les plus couramment étudiés, nous cherchons maintenant à identifier la relation optimale entre le temps de fonctionnement des algorithmes d'approximation, exprimé en fonction

de la taille de l'instance n, le paramètre de distance d et le meilleur ratio d'approximation réalisable ρ pour le problème, afin d'impliquer la taille de l'entrée et la qualité des solutions en jeu comme paramètres dans nos investigations.

Par la suite, notre perspective change en termes de types de résultats que nous explorons : dans les chapitres précédents où notre point de vue a été paramétré, notre objectif était d'identifier les valeurs correctes des temps de fonctionnement pour chaque paramètre considéré, en montrant que ceux-ci sont dans un certain sens optimaux en fournissant des limites supérieures et inférieures correspondantes, ou des résultats algorithmiques et de difficulté concurrents.

En ce qui concerne l'approximation, notre objectif reste d'assurer des limites étroites qui découlent de résultats algorithmiques et de difficulté concordants, mais ce sont maintenant les fonctions décrivant les ratios d'approximation que nous souhaitons caractériser précisément, par l'identification de limites supérieures/inférieures sur les ratios réalisables dans des temps de fonctionnement strictement polynomiaux. Celles-ci sont encore influencées par des restrictions à la structure de l'entrée, car pour nos résultats nous considérons des graphes de degrés bornés et des graphes bipartites, qui présentent un intérêt particulier en termes d'indépendance.

Avant de passer à la description de nos résultats, nous notons que ceux-ci peuvent dépendre de la parité de notre paramètre de distance d comme étant pair ou impair. Tant nos temps de parcours que nos ratios peuvent être affectés par cette particularité du problème qui, intuitivement, se pose en raison de la existence d'un sommet *moyen* sur un chemin de longueur d entre deux points terminaux.

Si d est pair, alors un tel sommet peut exister à une distance égale $d/2$ de n'importe quel nombre de sommets dans la solution, tandis que si d est impair, il ne peut y avoir de sommet à une distance égale de n'importe quelle paire de sommets dans la solution. Cette idiosyncrasie peut changer la façon dont nos algorithmes et nos constructions de difficulté fonctionnent (déjà vu dans l'algorithme du Théorème 56) et dans certains cas, elle modifie même entièrement la complexité du problème (par exemple dans les résultats de [42]).

Nos résultats: La section 5.1 concerne les temps de fonctionnement super-polynomiaux, en présentant d'abord un algorithme exponentiel exact pour d-Scattered Set de complexité $O^*((ed)^{\frac{2n}{d}})$ et en considérant ensuite l'inapproximabilité du problème dans la même plage. Nous montrons qu'aucun algorithme de ρ-approximation ne peut prendre du temps (approximativement) $2^{\frac{n^{1-\epsilon}}{\rho d}}$ pour les d pairs et $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ pour les d impairs, sous l'ETH (randomisé). Il est complété par des algorithmes de ρ-approximation (presque) identiques des temps de fonctionnement $O^*((e\rho d)^{\frac{2n}{\rho d}})$ pour les d pairs et $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ pour les d impairs.

Nous constatons que les PCP actuels[2] sont incapables de distinguer entre les temps de fonctionnement optimaux de la forme $2^{n/\rho}$ et $\rho^{n/\rho}$ pour les algorithmes d'approximation ρ, en raison du facteur poly-logarithmique ajouté par les constructions les plus efficaces et nous ne nous concentrons donc pas sur les facteurs poly-logarithmiques différenciant nos limites supérieure et inférieure.

Ces résultats fournissent une caractérisation complète de la relation optimale entre le ratio d'approximation le plus défavorable ρ réalisable pour d-Scattered Set par

[2]Versions du célèbre *théorème PCP* : intuitivement, cela se réfère à la machinerie (plutôt complexe) derrière les réductions introduisant les écarts les plus efficaces [95].

n'importe quel algorithme, son temps de fonctionnement et le paramètre de distance d, *pour n'importe quel point de la courbe d'arbitrage*, de la même manière que cela a été fait pour Independent Set dans [26, 36] (aussi [21, 23]), en considérant également la gamme des valeurs possibles pour d.

Nous observons que le paramètre de distance d agit comme un facteur d'échelle pour la taille de l'instance, le problème devenant plus facile lorsque les sommets doivent être beaucoup plus éloignés, une caractéristique contrebalancée par le ratio d'approximation choisi ρ, de petites valeurs garantissant la qualité des solutions produites, mais ayant également un impact négatif sur l'exposant du temps de fonctionnement.

La section 5.2 poursuit en considérant des temps de fonctionnement strictement polynomiaux. Nous montrons tout d'abord qu'il n'existe pas d'algorithme d'approximation polynomiale du temps pour d-Scattered Set avec le ratio $\Delta^{\lfloor d/2 \rfloor - \epsilon}$ dans les graphes de degré maximum Δ. C'est la première borne inférieure qui considère Δ et qui généralise l'inapproximabilité connue $\Delta^{1-\epsilon}$ de Independent Set (Théorème 5.2 de la [26], reformulé ici sous le nom de Théorème 10, ainsi que [3]).

Le degré maximal du sommet Δ a un rôle important dans le contexte de l'indépendance (par exemple, [8, 38, 57]) et a été spécifiquement étudié pour d-Scattered Set dans [43], où les approximations $O(\Delta^{d-1})$- et $O(\Delta^{d-2}/d)$ en temps polynomial sont données. Nous améliorons cette méthode en montrant que tout algorithme d'approximation gourmand atteint en fait un ratio de $O(\Delta^{\lfloor d/2 \rfloor})$, correspondant également à notre limite inférieure.

Enfin, nous nous tournons vers les graphes bipartites et montrons que d-Scattered Set peut être approché avec un facteur de $2\sqrt{n}$ en temps polynomial également pour les valeurs paires de d, correspondant à l'inapproximabilité connue de $n^{1/2-\epsilon}$ de [42] et complétant l'approximation connue de $\sqrt{n}$ pour les valeurs impaires de d de [56].

Nous clôturons le chapitre par une note supplémentaire sur le treewidth des graphes de puissance obtenus par des observations liées à nos résultats précédents. Ceux-ci sont également résumés dans le tableau 7.3 ci-dessous.

	Inapproximabilité	Approximation
Super-polynomial	$2^{\frac{n^{1-\epsilon}}{\rho d}}$ (66)/ $2^{\frac{n^{1-\epsilon}}{\rho(d+\rho)}}$ (67)	$O^*((e\rho d)^{\frac{2n}{\rho d}})$ (68)/ $O^*((e\rho d)^{\frac{2n}{\rho(d+\rho)}})$ (69)
Polynomial	$\Delta^{\lfloor d/2 \rfloor - \epsilon}$ (70)	$O(\Delta^{\lfloor d/2 \rfloor})$ (83)
Graphes bipartites	$n^{1/2-\epsilon}$ [42]	$2\sqrt{n}$ (86)

Table 7.3: Un résumé de nos résultats (nombres de théorème), pour des valeurs paires/impaires de d.

Conclusion

Nous avons déjà observé dans les deux cas que l'influence du paramètre de distance sur la complexité du problème n'est pas négligeable, car tant (k, r)-Center et d-Scattered Set deviennent plus difficiles que Dominating Set et Independent Set lorsque r ou d est illimité : nos résultats de W-difficulté pour la paramétrisation par k et vc (pondéré), fvs (non pondéré) nous dictent que si le paramètre de distance prend des valeurs importantes, même la structure (considérablement) restreinte du graphe ne sera pas d'un grand secours pour améliorer de manière significative les temps en cours d'exécution sous XP.

Considérant que ces deux problèmes ont déjà été prouvés insolubles par presque toutes les autres alternatives de calcul à un calcul exact (voir Section 2.2), tout en prenant également en compte spécifiquement pour d-Scattered Set, son $\Delta^{d/2}$-inapproximabilité en temps polynomial où d apparaît sur l'exposant du ratio, ainsi que le fait que pour des d assez importants, même le PTAS pour les graphes planaires de [43] n'est pas applicable (voir ci-dessous), il semble que dans les deux cas, une bonne façon d'aborder ces problèmes lorsque les paramètres de distance sont importants est d'appliquer soit nos algorithmes FPT pour le cas non pondéré paramétré par vc/td, soit nos schémas d'approximation FPT basés sur tw/cw dont les durées de fonctionnement dépendent uniquement de la structure du graphe d'entrée.

Il convient de noter que les deux requièrent d'importantes restrictions sur l'entrée, tandis que le premier est incompatible avec le cas pondéré important et que le second peut effectivement identifier correctement la taille de la solution optimale, bien qu'avec une perte de précision admissible en termes de satisfaction des exigences de distance.

Problèmes en suspens: Étant donné l'intraçabilité et l'inapproximabilité des paramétrages standard des deux problèmes, ainsi que les limites inférieures de leur complexité structurellement paramétrée basées sur la SETH, nous nous intéressons ensuite aux meilleurs ratios réalisables par des algorithmes d'approximation structurellement paramétrés ou des temps d'exécution qui améliorent clairement les limites inférieures des cas exacts donnés ici. À titre d'exemple concret, quel serait le meilleur ratio réalisable ρ par un algorithme d'approximation à 2^{tw}-temps pour l'un ou l'autre problème où la valeur de r/d est illimitée ?

En dehors de cette direction et compte tenu également des travaux antérieurs connexes (notamment [22]), les résultats ci-dessus peuvent être considérés comme complétant le tableau sur la complexité structurellement paramétrée du problème (k, r)-Center. Néanmoins, certaines questions restent en suspens concernant l'affinement de nos limites inférieures basées sur l'ETH en utilisant comme point de départ la plus précise SETH, ainsi que le statut de complexité du problème par rapport à d'autres paramètres structurels, tels que rankwidth, ou neighborhood diversityla largeur modulaire ou la diversité des quartiers.

Les questions qui restent ouvertes sur la complexité du problème d-Scattered Set en termes de paramètres structurels peuvent concerner l'identification de limites supérieures et inférieures (basées sur la SETH) similaires pour clique-width (FPT pour $d = 2$), ainsi que l'affinement de nos limites inférieures basées sur l'ETH pour vc et fvs, qui ne sont pas considérées comme strictes en raison de l'augmentation quadratique de la taille des paramètres dans nos réductions.

Sur notre travail (super-)polynomial sur d-Scattered Set, à part de la possibilité de "dé-randomisation" des résultats ci-dessus qui utilisent la construction randomisée de [26] comme point de départ, certaines questions sans réponse impliquent :

- l'affinement de nos limites supérieures super-polynomiales pour qu'elles correspondent exactement aux limites inférieures, c'est-à-dire ρ-approximations dans le temps $O^*(2^{\frac{2n}{\rho d}})$ pour les d pairs et $O^*(2^{\frac{2n}{\rho(d+\rho)}})$ pour les d impairs, notant que même les algorithmes de temps de fonctionnement conservant le même exposant mais dont la base ne dépend pas de d laisseraient entrevoir que le problème devient en fait plus facile à approximer pour des valeurs assez importantes de d;

- la complexité du problème sur les graphes bipartites chordales, également mentionné comme un problème ouvert par [42];
- la fonctionnalité du PTAS pour les graphes planaires des mêmes auteurs, qui ne fonctionne que pour des valeurs fixes de d, car elle étend l'approche bien connue de [6] pour obtenir de tels algorithmes pour plusieurs problèmes, notamment INDEPENDENT SET.

Comme la technique de [6] consiste à décomposer le graphe en sous-graphes d-outerplanaires (approximativement) puis à résoudre exactement le problème dans chacun d'eux en utilisant une programmation dynamique sur leurs décompositions d'arbre, pour des valeurs de d qui ne sont pas constantes (disons $d \geq \sqrt{n}$)), cela n'est pas réalisable en temps polynomial en raison de l'exposant des algorithmes de treewidth dépendant de d.

Il serait intéressant de voir une extension de cette approche (ou d'une autre) pour le cas des d non limités, ou, à l'inverse, une réduction de difficulté prouvant qu'elle est peu probable. La partie difficile ici devrait impliquer une construction très efficace en termes de gadgets *crossing* afin de maintenir la planéité, ou, de l'autre côté, un moyen de résoudre le problème de manière optimale dans des sous-graphes soigneusement construits sans l'exigence exponentielle de d. Notez que cela est intuitivement lié à la question de savoir si le problème devient plus facile à approximer avec de gros d, une question également soulevée par notre premier point ci-dessus.

Bibliography

[1] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[2] J. Alber and R. Niedermeier. Improved Tree Decomposition Based Algorithms for Domination-like Problems. In *LATIN*, volume 2286 of *LNCS*, pages 613–628, 2002.

[3] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. *Computational Complexity*, 5(1):60–75, 1995.

[4] E. Angel, E. Bampis, B. Escoffier, and M. Lampis. Parameterized Power Vertex Cover. In *WG*, volume 9941 of *LNCS*, pages 97–108, 2016.

[5] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1st edition, 2009.

[6] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[7] J. Bar-Ilan, G. Kortsarz, and D. Peleg. How to Allocate Network Centers. *Journal of Algorithms*, 15(3):385–415, 1993.

[8] P. Berman and M. Karpinski. On some tighter inapproximability results. In *ICALP*, volume 1644 of *LNCS*, pages 200–209, 1999.

[9] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: fast subset convolution. In *STOC 2007*, pages 67–74. ACM.

[10] H. L. Bodlaender. The algorithmic theory of treewidth. *Electronic Notes in Discrete Mathematics*, 5:27–30, 2000.

[11] H. L. Bodlaender. Treewidth: Characterizations, Applications, and Computations. In *WG*, volume 4271 of *LNCS*, pages 1–14, 2006.

[12] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[13] H. L. Bodlaender and T. Hagerup. Parallel Algorithms with Optimal Speedup for Bounded Treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998.

[14] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *Computer Journal*, 51(3):255–269, 2008.

[15] H. L. Bodlaender, E. J. van Leeuwen, J. M. M. van Rooij, and M. Vatshelle. Faster Algorithms on Branch and Clique Decompositions. In *MFCS*, volume 6281 of *LNCS*, pages 174–185, 2010.

[16] B. Bollobás. *Modern Graph Theory*. Graduate texts in mathematics. Springer, 1998.

[17] B. Bollobás and F. R. K. Chung. The diameter of a cycle plus a random matching. *SIAM Journal on Discrete Mathematics*, 1(3):328–333, 1988.

[18] B. Bollobás and W. F. de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125–134, 1982.

[19] J. A. Bondy. *Graph Theory With Applications*. Elsevier Science Ltd., 1976.

[20] E. Bonnet, B. Escoffier, E. J. Kim, and V. T. Paschos. On subexponential and FPT-time inapproximability. *Algorithmica*, 71(3):541–565, Mar. 2015.

[21] E. Bonnet, M. Lampis, and V. Paschos. Time-Approximation Trade-offs for Inapproximable Problems. In *STACS*, volume 47 of *LIPIcs*, page 22:1–22:14, 2016.

[22] G. Borradaile and H. Le. Optimal Dynamic Program for r-Domination Problems over Tree Decompositions. In *IPEC*, volume 63 of *LIPIcs*, pages 8:1–8:23, 2016.

[23] N. Bourgeois, B. Escoffier, and V. T. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discrete Applied Mathematics*, 159(17):1954–1970, 2011.

[24] A. Brandstädt and F. F. Dragan. A linear-time algorithm for connected r-domination and Steiner tree on distance-hereditary graphs. *Networks*, 31(3):177–182, 1998.

[25] P. Chalermsook, M. Cygan, G. Kortsarz, B. Laekhanukit, P. Manurangsi, D. Nanongkai, and L. Trevisan. From gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *FOCS 2017*, pages 743–754. IEEE.

[26] P. Chalermsook, B. Laekhanukit, and D. Nanongkai. Independent Set, Induced Matching, and Pricing: Connections and Tight (Subexponential Time) Approximation Hardnesses. In *FOCS 2013*, pages 370–379. IEEE.

[27] P. Chalermsook, B. Laekhanukit, and D. Nanongkai. Coloring Graph Powers: Graph Product Bounds and Hardness of Approximation. In *LATIN*, volume 8392 of *LNCS*, page 409–420, 2014.

[28] S. Chechik and D. Peleg. The fault-tolerant capacitated K-center problem. *Theoretical Computer Science*, 566:12–25, 2015.

[29] Y. Chen and B. Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM Journal on Computing*, 48(2):513–533, 2019.

[30] R. S. Coelho, P. F. S. Moura, and Y. Wakabayashi. The k-hop connected dominating set problem: hardness and polyhedra. *Electronic Notes in Discrete Mathematics*, 50:59–64, 2015.

[31] M. B. Cohen. Ramanujan graphs in polynomial time. In *FOCS 2016*, pages 276–281. IEEE.

[32] B. Courcelle. On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Mathematics*, 160(6):866 – 887, 2012.

[33] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[34] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

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

[36] M. Cygan, L. Kowalik, M. Pilipczuk, and M. Wykurz. Exponential-time approximation of hard problems. *CoRR*, abs/0810.4934, 2008.

[37] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

[38] M. Demange and V. T. Paschos. Improved approximations for maximum independent set via approximation chains. *Applied Mathematics Letters*, 10(3):105 – 110, 1997.

[39] R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[40] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[41] D. Eisenstat, P. N. Klein, and C. Mathieu. Approximating k-center in planar graphs. In *SODA 2014*, pages 617–627. SIAM.

[42] H. Eto, F. Guo, and E. Miyano. Distance-d independent set problems for bipartite and chordal graphs. *Journal of Combinatorial Optimization*, 27(1):88–99, 2014.

[43] H. Eto, T. Ito, Z. Liu, and E. Miyano. Approximability of the Distance Independent Set Problem on Regular Graphs and Planar Graphs. In *COCOA*, volume 10043 of *LNCS*, pages 270–284, 2016.

[44] H. Eto, T. Ito, Z. Liu, and E. Miyano. Approximation Algorithm for the Distance-3 Independent Set Problem on Cubic Graphs. In *WALCOM*, volume 10167 of *LNCS*, pages 228–240, 2017.

[45] T. Feder and D. H. Greene. Optimal Algorithms for Approximate Clustering. In *STOC 1988*, pages 434–444. ACM.

[46] A. Feldmann. Fixed Parameter Approximations for k-Center Problems in Low Highway Dimension Graphs. In *ICALP*, volume 9135 of *LNCS*, 2015.

[47] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. 2006.

[48] F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Bidimensionality and EPTAS. In *SODA 2011*, pages 748–759. SIAM.

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

[50] V. Garnero, C. Paul, I. Sau, and D. M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM Journal on Discrete Mathematics*, 29:1864–1894, 2015.

[51] P. Golovach and Y. Villanger. Parameterized Complexity for Domination Problems on Degenerate Graphs. In *WG*, volume 5344 of *LNCS*, pages 195–205, 2008.

[52] T. F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

[53] F. Gurski. A comparison of two approaches for polynomial time algorithms computing basic graph parameters. *CoRR*, abs/0806.4073, 2008.

[54] F. Gurski and E. Wanke. The Tree-Width of Clique-Width Bounded Graphs Without $K_{n,n}$. In *WG*, volume 1928 of *LNCS*, pages 196–205, 2000.

[55] M. T. Hajiaghayi, R. Khandekar, and G. Kortsarz. The foundations of fixed parameter inapproximability. *CoRR*, abs/1310.2711, 2013.

[56] M. M. Halldorsson, J. Kratochvil, and J. A. Telle. Independent Sets with Domination Constraints. *Discrete Applied Mathematics*, 99(1-3):39–54, 2000.

[57] M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, May 1997.

[58] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(38), 1997.

[59] D. S. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

[60] R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[61] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[62] I. Katsikarelis, M. Lampis, and V. T. Paschos. Structural Parameters, Tight Bounds, and Approximation for (k, r)-Center. *CoRR*, abs/1704.08868, 2017.

[63] I. Katsikarelis, M. Lampis, and V. T. Paschos. Structural Parameters, Tight Bounds, and Approximation for (k, r)-Center. In *ISAAC*, volume 92 of *LIPIcs*, pages 50:1–50:13, 2017.

[64] I. Katsikarelis, M. Lampis, and V. T. Paschos. Structurally parameterized d-scattered set. *CoRR*, abs/1709.02180, 2017.

[65] I. Katsikarelis, M. Lampis, and V. T. Paschos. Improved (in-)approximability bounds for d-scattered set. In *WAOA*, volume 11926 of *LNCS*, pages 202–216, 2019.

[66] I. Katsikarelis, M. Lampis, and V. T. Paschos. Improved (in-)approximability bounds for d-scattered set. *CoRR*, abs/1910.05589, 2019.

[67] I. Katsikarelis, M. Lampis, and V. T. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discrete Applied Mathematics*, 264:90 – 117, 2019. Combinatorial Optimization: between Practice and Theory.

[68] I. Katsikarelis, M. Lampis, and V. Th. Paschos. Structurally parameterized d-scattered set. In *WG*, volume 11159 of *LNCS*, pages 292–305, 2018.

[69] S. Khuller, R. Pless, and Y. J. Sussmann. Fault tolerant K-center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000.

[70] S. Khuller and Y. Sussmann. The Capacitated K-Center Problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

[71] T. Kloks. *Treewidth, Computations and Approximations*. Springer, 1994.

[72] S. O. Krumke. On a Generalization of the p-Center Problem. *Information Processing Letters*, 56(2):67–71, 1995.

[73] M. Lampis. Parameterized Approximation Schemes Using Graph Widths. In *ICALP*, volume 8572 of *LNCS*, pages 775–786, 2014.

[74] M. Lampis, K. Makino, V. Mitsou, and Y. Uno. Parameterized Edge Hamiltonicity. In *WG*, volume 8747 of *LNCS*, pages 348–359, 2014.

[75] A. Leitert and F. Dragan. Parameterized Approximation Algorithms for some Location Problems in Graphs. *CoRR*, abs/1706.07475v1, 2017.

[76] D. Lokshtanov, D. Marx, and S. Saurabh. Known Algorithms on Graphs on Bounded Treewidth are Probably Optimal. In *SODA 2011*, pages 777–789. SIAM.

[77] D. Lokshtanov, N. Misra, G. Philip, M. S. Ramanujan, and S. Saurabh. Hardness of r-dominating set on Graphs of Diameter $(r + 1)$. In *IPEC*, volume 8246 of *LNCS*, pages 255–267, 2013.

[78] D. Marx. Efficient Approximation Schemes for Geometric Problems? In *ESA*, volume 3669 of *LNCS*, pages 448–459, 2005.

[79] D. Marx. Parameterized Complexity and Approximation Algorithms. *Computer Journal*, 51(1):60–78, 2008.

[80] D. Marx and M. Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *ESA*, volume 9294 of *LNCS*, 2015.

[81] P. Montealegre and I. Todinca. On Distance-d Independent Set and other problems in graphs with few minimal separators. In *WG*, volume 9941 of *LNCS*, page 183–194, 2016.

[82] D. Moshkovitz. The Projection Games Conjecture and the NP-Hardness of $\ln n$-Approximating Set-Cover. *Theory of Computing*, 11:221–235, 2015.

[83] J. Nesetril and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

[84] S. Oum, S. H. Sæther, and M. Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theoretical Computer Science*, 535:16–24, 2014.

[85] R. Panigrahy and S. Vishwanathan. An $O(log * n)$-Approximation Algorithm for the Asymmetric p-Center Problem. *Journal of Algorithms*, 27(2):259–268, 1998.

[86] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425 – 440, 1991.

[87] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[88] M. Pilipczuk and S. Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *CoRR*, abs/1809.05675, 2018.

[89] K. C. S., B. Laekhanukit, and P. Manurangsi. On the parameterized complexity of approximating dominating set. STOC 2018. ACM.

[90] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[91] P. J. Slater. R-Domination in Graphs. *Journal of the ACM*, 23(3):446–450, 1976.

[92] A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995.

[93] J. A. Telle and A. Proskurowski. Practical Algorithms on Partial k-Trees with an Application to Domination-like Problems. In *WADS*, volume 709 of *LNCS*, pages 610–621, 1993.

[94] J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *ESA*, volume 5757 of *LNCS*, pages 566–577, 2009.

[95] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[96] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 1st edition, 2011.

[97] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Electronic Colloquium on Computational Complexity (ECCC)*, 2005.

RÉSUMÉ

Nous nous concentrons sur les problèmes (k, r)-CENTER et d-SCATTERED SET qui généralisent les concepts de *domination* et *indépendance* des sommets, sur les distances plus grandes.
Dans la première partie, nous examinons le paramétrage standard, ainsi que les paramètres des graphes mesurant la structure de l'entrée. Nous proposons des résultats qui montrent qu'il n'existe pas d'algorithme avec un temps d'exécution inférieur à certaines limites, si l'hypothèse du temps exponentiel est vraie, nous produisons des algorithmes de complexité essentiellement optimale qui correspondent à ces limites et nous essayons en outre de proposer une alternative au calcul exact en temps considérablement réduit, grâce à l'approximation.
Dans la deuxième partie, nous considérons l'approximabilité (super-)polynômiale du problème de d-SCATTERED SET c'est-à-dire que nous déterminons la relation exacte entre un réalisable rapport d'approximation ρ, le paramètre de distance d et le temps d'exécution de l'algorithme avec un rapport de ρ, en fonction de ce qui précède et de la taille de l'entrée n. Nous considérons ensuite les temps d'exécution strictement polynomiaux et les graphes de degré maximal borné, ainsi que les graphes bipartites.

MOTS CLÉS

Paramétrage, approximation, indépendance, domination, ETH, SETH, paramètre de distance

ABSTRACT

In this thesis we focus on the NP-hard problems (k, r)-CENTER and d-SCATTERED SET that generalize the well-studied concepts of *domination* and *independence* over larger distances.
In the first part we maintain a parameterized viewpoint and examine the standard parameterization as well as the most widely-used graph parameters measuring the input's structure. We offer hardness results that show there is no algorithm of running-time below certain bounds, subject to the (Strong) Exponential Time Hypothesis, produce essentially optimal algorithms of complexity that matches these lower bounds and further attempt to offer an alternative to exact computation in significantly reduced running-time by way of approximation algorithms.
In the second part we consider the (super-)polynomial (in-)approximability of the d-SCATTERED SET problem, i.e. we determine the exact relationship between an achievable approximation ratio ρ, the distance parameter d, and the running-time of any ρ-approximation algorithm expressed as a function of the above and the size of the input n. We then consider strictly polynomial running-times and improve our understanding on the approximability characteristics of the problem on graphs of bounded maximum degree as well as bipartite graphs.

KEYWORDS

Parameterization, Approximation, Independence, Domination, ETH, SETH, treewidth, clique-width, distance parameter, vertex degree