Discrete Mathematics and Theoretical Computer Science DMTCS vol. **17:2**, 2015, 77–100

Improving Vertex Cover as a Graph Parameter

Robert Ganian*

TU Wien, Vienna, Austria

received 1st July 2013, revised 21st Nov. 2014, 12th May 2015, 27th May 2015, accepted 7th Aug. 2015.

Parameterized algorithms are often used to efficiently solve NP-hard problems on graphs. In this context, vertex cover is used as a powerful parameter for dealing with graph problems which are hard to solve even when parameterized by tree-width; however, the drawback of vertex cover is that bounding it severely restricts admissible graph classes. We introduce a generalization of vertex cover called twin-cover and show that FPT algorithms exist for a wide range of difficult problems when parameterized by twin-cover. The advantage of twin-cover over vertex cover is that it imposes a lesser restriction on the graph structure and attains low values even on dense graphs.

Apart from introducing the parameter itself, this article provides a number of new FPT algorithms parameterized by twin-cover with a special emphasis on solving problems which are not in FPT even when parameterized by tree-width. It also shows that MS_1 model checking can be done in elementary FPT time parameterized by twin-cover and discusses the field of kernelization.

Keywords: Vertex cover, Twin-cover, Parameterized Complexity

1 Introduction

Parameterized complexity is a very useful approach for dealing with NP-hard graph problems. In real-life applications it is usually not necessary to solve problems on general graphs, but rather on graphs with some kind of structure present. It is then possible to use a structural parameter k to describe this structure and use it to design efficient algorithms as long as k is small. Specifically, we are mainly interested in so-called Fixed Parameter Tractable algorithms (FPT algorithms in short, see Subsection 2.1).

It is a well-known fact that FPT algorithms exist for a large number of NP-hard problems parameterized by tree-width. However, more powerful parameters are required to deal with problems which are not likely to admit FPT algorithms when parameterized by tree-width, and vertex cover is arguably the most popular choice in this regard. A wide range of such problems have been solved in FPT time parameterized by vertex cover, for instance EQUITABLE COLORING [FGK11], EQUITABLE CONNECTED PARTITION [EFG+09], BOXICITY [ACS10, FHR12], PRECOLORING EXTENSION [FGK11], and various graph layout problems such as IMBALANCE, CUTWIDTH, DISTORTION and BANDWIDTH [FLM+08]. The general MS_1 MODEL CHECKING problem has also been shown to admit an elementary FPT algorithm when parameterized by vertex cover [Lam12].

*The author acknowledges support by the FWF Austrian Science Fund (X-TRACT, P26696) and by the Czech Science Foundation, project P202/11/0196. An extended abstract of this paper was presented at IPEC 2011 [Gan11].

But if vertex cover is so powerful in parameterized algorithmics, why even bother with other parameters at all? The answer is that any good structural parameter needs to balance between being as powerful as possible (i.e. useful in designing algorithms) while also being as little restrictive as possible (the class of graphs with the parameter bounded should be rich). The greatest disadvantage of vertex cover is that it is very restrictive, and this severely limits its usefulness in practical applicatons.

The article introduces twin-cover as a direct generalization of vertex cover towards richer graph classes, including dense graph classes (which have unbounded tree-width). Aside from providing new structural results for twin-cover, we also include a comparison to the measure of neighborhood diversity which was recently introduced by Lampis [Lam12].

We show that efficient algorithms exist on graphs of bounded twin-cover for a very wide range of problems of practical interest, some of which are W[1]-hard when parameterized by tree-width or even NP-hard for constant values of tree-width. In fact, the contribution of our parameter is twofold: it directly generalizes several of the best known FPT algorithms for problems where vertex cover previously had to be used, and at the same time provides a means of efficiently solving problems which do not admit FPT algorithms when parameterized by known graph parameters which are suitable for dense graphs (such as clique-width or rank-width).

Preliminary notions are introduced in Section 2, and Section 3 then discusses the motivation and background of twin-cover. Section 4 contains new FPT algorithms for problems which are not in FPT when parameterized by tree-width. The problems we solve here are GRAPH MOTIF, EQUITABLE COLORING, PRECOLORING EXTENSION, BOXICITY, and FIREFIGHTING. Section 5 introduces two FPT algorithms for problems which are W[1]-hard when parameterized by clique-width, and also discusses the field of kernelization. Section 6 proves that Monadic Second-Order logic can be model checked in elementary FPT time parameterized by twin-cover, and Section 7 contains the concluding notes and discussion.

2 Preliminaries

A *graph* is a pair $G = (V, E)$ such that $E \subseteq [V]^2$, i.e. the elements of E are 2-element subsets of V. The elements of V are called *vertices* and the elements of E are *edges*. It is always assumed that $V \cap E = \emptyset$. The vertex set of a graph G is referred to as $V(G)$, and the analogous holds for $E(G)$. The graphs we consider in this paper are *simple*, i.e. they do not contain loops and parallel edges. For brevity, we will often refer to an edge $\{a, b\} \in E$ simply as ab.

A vertex v is *incident* with an edge ab if $v \in \{a, b\}$. We say two vertices $x, y \in V(G)$ are *adjacent*, or *neighbors*, if $xy \in E(G)$. The *degree* of a vertex v is the number of edges which contain v, and is equal to the number of neighbors of v. The neighborhood of a vertex v is the set of its neighbors. If all the vertices of G are pairwise adjacent, then G is *complete*, and we use K_n to refer to complete graphs with n vertices. Vertices t, u are called *twins* if all other vertices are adjacent either to both t, u, or none (or, equivalently, if t and u have the same neighborhood except for each other). If t, u are twins and are adjacent to each other, then we call them *true twins* and the edge tu is called a *twin edge*.

A graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Additionally, if G' contains all edges $xy \in E$ with $x, y \in V'$ then G' is an induced subgraph of G. A non-empty graph G is called *connected* if, for any partition of the vertex set into a nonempty V_1 and a nonempty V_2, there exists an edge between at least one $a \in V_1$ and one $b \in V_2$. A maximal connected subgraph of G is called a *connected component* of G. The *complement* G' of $G = (V, E)$ is the graph $(V, [V]^2 \setminus E)$.

A *clique* K in a graph G is a subgraph of G such that there is an edge between each pair of distinct vertices in K. A *path* P in a graph G is a subgraph of G which is connected, contains two vertices of degree 1 (the endpoints) and all remaining vertices in P have degree 2.

A *vertex cover* of a graph $G = (V, E)$ is a subset $X \subseteq V$ such that each edge of G is incident to at least one $x \in X$. The *vertex cover number* is then the size of a minimum-cardinality vertex cover in G. For a fixed vertex cover X, we say that a vertex not in X is a *non-cover vertex*, and a *type* $T \subseteq V$ is an inclusion-maximal set of vertices such that all vertices in a type T have the same neighborhood in X and $T \cap X = \emptyset$. Notice that T is a set of independent vertices which are twins, and that the number of types is bounded by $2^{|X|}$.

A more detailed overview of graph theory may be found, e.g., in [Die05].

2.1 Parameterized complexity

We adopt the standard notation introduced by Downey and Fellows [DF99], as it is used in various publications and surveys such as [Bod09].

A *parameterized problem* is a subset of $\Sigma^* \times \mathbb{N}$ for some fixed alphabet Σ. In other words, we look at decision problems where some specific part of the input, called the *parameter*, is an integer.

The theory of fixed parameter complexity is used to distinguish between the running time of algorithms for parameterized problems, where we pay attention how this time depends on the parameter as well as on the input size. Three important types of behavior can be observed:

- **paraNP-complete**: The problem is NP-complete for some fixed values of the parameter. For instance, GRAPH COLORING is NP-complete even when the number of colors is 3.
- **XP**: For every fixed value of the parameter k, there exists a polynomial-time algorithm solving the problem on inputs of size n, but the exponent of the running time grows with k, i.e. the running time is $n^{f(k)}$ for some function f.
- **FPT**: There is an algorithm that solves the problem in time $f(k)n^c$ for some function f on inputs of size n with parameter k, where c is a constant. Such an algorithm is called an FPT algorithm, and FPT is defined as the class of all parameterized problems that have such an algorithm. FPT is short for *fixed parameter tractable*.

The main benefit of FPT algorithms is that they allow efficient solution of otherwise hard problems on inputs of large sizes as long as their parameter is sufficiently low. For hard problems, the ideal case is for f to be at most a single-exponential function, however sometimes double- or higher-exponential functions are necessary.

Downey and Fellows also defined the complexity class W[1] along with a number of other complexity classes of parameterized problems. For the precise definition of W[1], please see, e.g., [DF99]. While FPT $\subseteq$ W[1], it is widely believed that FPT $\neq$ W[1], and so parameterized problems which are hard for W[1] (under an appropriate defined parameterized reduction) are not believed to be in FPT.

We will also briefly deal with the closely related field of kernelization, which studies the efficient preprocessing of inputs. A *kernelization algorithm* for a parameterized problem P is an algorithm A that transforms inputs (I, k) of P to inputs (I', k') of P, such that:

1. the algorithm uses time polynomial in $|I| + k$;
2. the algorithm transforms inputs to equivalent inputs: $(I, k) \in P \iff A(I, k) \in P$;

3. $k' \le k$;
4. $|I'| \le f(k)$ for some function f: the size of the new input is bounded by a function of the value of the old parameter.

We then say that P has a *kernel of size* f, and specifically that P has a *polynomial kernel* if f is polynomial. More information about kernelization may be found, e.g., in the survey by Bodlaender [Bod09].

It is a well-known fact that any parameterized problem P belongs to the class FPT if and only if P has a kernel and is decidable. However, this is not true for polynomial kernels, as there exist parameterized problems in FPT which are believed not to admit a polynomial kernel [BDFH09].

3 A generalization of vertex cover

It has already been said that the usefulness of vertex cover in practice is severely limited by how restrictive it is. Also recall that one of the main reasons for using vertex cover as a parameter is to solve problems which are not in FPT when parameterized by tree-width (such problems are frequently also not in FPT when parameterized by path-width, and some are even NP-hard on trees). Yet there exist graph classes with unbounded vertex cover where virtually all our problems of interest are easily solvable, such as complete graphs. This suggests that there should exist a parameter which

1. attains low values on a significantly more general class of graphs, including cliques and graphs of bounded vertex cover, and
2. is capable of solving "most" problems which are solvable by vertex cover.

Note that the sought parameter cannot solve *all* of these problems, since Courcelle, Makowski and Rotics have shown the existence of problems which are hard on complete graphs and yet solvable in FPT time when parameterized by tree-width [CMR00]. To be more precise, there exist MS_2-definable graph problems which cannot be solved on cliques in polynomial time, unless $EXP = NEXP$. However, known problems of this kind are of mostly theoretical interest and have little practical importance.

3.1 Neighborhood diversity

Neighborhood diversity was introduced in [Lam12] as a useful tool for dealing with Monadic Second-Order logic. The basic idea behind the notion stems from the following structure (recall that $N(v)$ denotes the neighborhood of v):

Definition 1 ([Lam12]) *We will say that two vertices v, v_0 of a graph $G = (V, E)$ have the same* neighborhood-type *iff* $N(v) \backslash \{v_0\} = N(v_0) \backslash \{v\}$.

Definition 2 (Definition 2 of [Lam12]) *A graph $G = (V, E)$ has neighborhood diversity at most w, if there exists a partition of V into at most w sets, such that all the vertices in each set have the same neighborhood-type.*

Note that our Definition 2 differs slightly from the definition given in [Lam12] – while the latter also considers colored graphs, the former restricts itself to uncolored (or single-colored) graphs. We refer to Subsection 3.4 for a comparison of neighborhood diversity to our proposed alternative.

3.2 Twin-cover

We propose another natural way of generalizing vertex cover to dense graphs. Instead of using the neighborhood structure, we relax the definition of vertex cover so that not all edges need to be covered.

Definition 3 *A set of vertices $X \subseteq V(G)$ is a twin-cover of G if for every edge $e = ab \in E(G)$ either*

1. *$a \in X$ or $b \in X$, or*
2. *a and b are true twins.*

We then say that G has twin-cover k if the size of a minimum twin-cover of G is k.

An illustration and comparison is provided in Figure 1. We continue with a few simple observations:

Observation 1

1. *The relation of "being true twins" is an equivalence, and equivalence classes of this relation form disjoint cliques in a graph.*
2. *If a graph does not contain any twin edges, then its vertex cover number and twin-cover number are identical.*
3. *The graph G' obtained by removing a twin-cover from G is a disjoint collection of cliques.*

Analogously to vertex cover, we will also use the notion of *types* with respect to twin-cover. Unlike the case of vertex cover, here each type is a disjoint union of complete graphs (or, equivalently, a collection of independent cliques).

We will use the following Lemma 2 to later show how to efficiently compute a minimum twin-cover (cf. Theorem 4). Note that Lemma 2 does not imply that finding a minimum vertex cover is in P if a minimum twin-cover is provided – in fact, a minimum vertex cover may not intersect with a minimum twin-cover at all (e.g., in a complete graph with one missing edge).

Lemma 2 *Given a graph G and a graph G' obtained by deleting all twin edges in G, the vertex cover number of G' is equal to the twin-cover number of G.*

Proof: Assume we are given a minimum vertex cover X' in G'. Then any edge in G is either covered by X' or is a twin edge, which does not have to be covered by the twin-cover. Thus X' is a twin-cover of G. On the other hand, assume we are given a minimum twin-cover X in G. Then X is also a vertex cover in G', since all edges in G which were not incident with X were deleted. □

Notice that complete graphs have a twin-cover of zero. Recall that, given a graph G with a twin-cover X, the connected components of $G - X$ form cliques. We refer to these cliques (including cliques of size 1) as the *non-cover cliques*.

Section 7 discusses an alternative – less restrictive – approach to defining a similar parameter, and why it fails. We also mention that there exists another, perhaps more intuitive definition for twin-cover:

Definition 4 *A set of vertices $X \subseteq V(G)$ is a twin-cover of G if there exists a subgraph G' of G such that*

1. *$X \subseteq V(G')$ and X is a vertex cover of G'.*

2. *G can be obtained by iteratively adding true twins to non-cover vertices in G'.*

Claim 1 *Definition 3 and Definition 4 are equivalent.*

Proof: Consider some set X which fulfills the conditions of Definition 3. To obtain G' as per Definition 4, it suffices to replace each clique in $G - X$ by a single vertex. It is easy to verify that X and G' satisfy the conditions of Definition 4.

On the other hand, consider some set X and subgraph G' which fulfill the conditions of Definition 4. Then X immediately satisfies the conditions in Definition 3. □

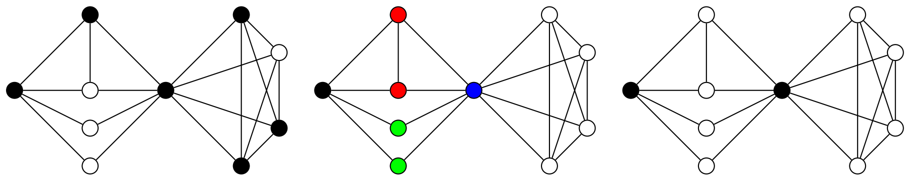

Fig. 1: From left to right: a minimum-size vertex cover (size 6 – *depicted in black*), the neighborhood diversity (size 5 – *each neighborhood-type with a different color*) and a minimum-size twin-cover (size 2 – *depicted in black*) of an example graph.

3.3 Definition of standard graph parameters

Here we define the graph parameters *tree-width* [RS90], *clique-width* [CMR00], *rank-width* [OS06, GH10] and *linear rank-width* [Gan10]. These are used mainly in the context of Proposition 3 later on.

Tree-width A *tree-decomposition* of a graph $G = (V, E)$ is a tree T over nodes $X_1, \dots X_n$ such that each node X_i in T is a subset of V and the following three conditions hold:

1. $V(G) = \bigcup_{X_i \in T} X_i$;
2. for every edge $ab \in E$ there exists some $X_i \in T$ such that $\{a, b\} \subseteq X_i$;
3. $X_a \cap X_c \subseteq X_b$ whenever $X_a, X_b, X_c \in T$ and X_b is on the unique path between X_a and X_c in T.

The width of T is the size of the largest set X_i minus one. The tree-width of G is the minimum width among all possible tree-decompositions of G.

Clique-width The clique-width of G is the minimum number of labels needed to construct G by the following 4 operations:

1. Creation of a new vertex v with label i,
2. Disjoint union of two labeled graphs G and H,
3. Addition of an edge between every vertex labeled by some label i and some label j, where $i \neq j$,
4. Renaming of (all occurrences of) label i to label j.

A sequence of such operations used to construct G with k labels is called a k-expression.

Rank-width For a graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U,W]$ denote the $U \times W$-submatrix of the adjacency matrix over the two-element field $\mathrm{GF}(2)$, i.e., the entry $a_{u,w}$, where $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U,W]$ is 1 if and only if uw is an edge of G. The *cut-rank* function ρ_G of a graph G is defined as follows: For a bipartition (U,W) of the vertex set $V(G)$, $\rho_G(U) = \rho_G(W)$ equals the rank of $\boldsymbol{A}_G[U,W]$ over $\mathrm{GF}(2)$.

A *rank-decomposition* of a graph G is a pair (T,μ) where T is a tree of maximum degree 3 and $\mu : V(G) \to \{t : t \text{ is a leaf of } T\}$ is a bijective function. For an edge e of T, the connected components of $T-e$ induce a bipartition (X,Y) of the set of leaves of T. The *width* of an edge e of a rank-decomposition (T,μ) is $\rho_G(\mu^{-1}(X))$. The *width* of (T,μ) is the maximum width over all edges of T. The rank-width of G is the minimum width over all rank-decompositions of G.

We call a rank-decomposition (T,μ) *linear* if T is a caterpillar, i.e., its degree-3 nodes form a path in T. The linear rank-width of G is then the minimum width over all rank-decompositions of G which are linear.

3.4 Twin-cover properties and comparison

There are two major problems with using neighborhood diversity as a generalization for vertex cover. First, the neighborhood diversity of a graph may be exponentially higher than its vertex cover, and so there are inputs on which the runtime guarantee provided by a single-exponential algorithm parameterized by neighborhood diversity is much worse than that of a single-exponential algorithm parameterized by vertex cover.

Second, many of the algorithms parameterized by vertex cover strongly rely on having a bounded number of "cover" vertices. Such algorithms would need to be redesigned from scratch to work on neighborhood diversity, and it is currently not clear how/whether this can be done.

On the other hand, twin-cover does not suffer from the problems listed above. Any vertex cover of a graph is also a viable twin-cover, and so the size of a minimum twin-cover is strictly less or equal to the minimum vertex cover. Additionally, we still have a small set of cover vertices whose removal trivializes the structure of the remainder of our graph. This allows the almost seamless adaptation of many algorithms parameterized by vertex cover.

That being said, it is important to note that the classes of graphs of bounded neighborhood diversity and/or bounded twin-cover are incomparable. As shown in Figure 2, there exist graph classes where one measure is bounded and the other is not. We provide a range of results on comparing twin-cover to various other graph parameters in Proposition 3.

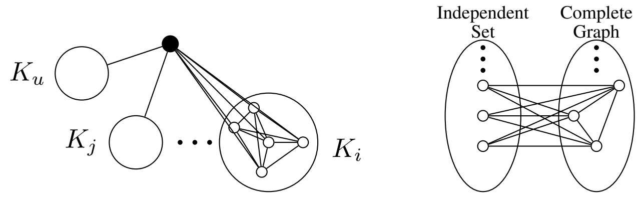

Fig. 2: Examples of graphs with unbounded neighborhood diversity and bounded twin-cover (left), and vice-versa (right). The left graph consists of a set of independent and disjoint cliques, all adjacent to a single vertex. The right graph is a complete bipartite graph where one side of the bipartition is a clique and the other is an independent set.

Proposition 3

1. *The vertex cover number of graphs of bounded twin-cover may be arbitrarily large.*
2. *There exist graphs with arbitrarily large twin-cover and bounded tree-width, and vice-versa.*
3. *The clique-width of graphs of twin-cover k is at most $k + 3$.*
4. *The rank-width and linear rank-width of graphs of twin-cover k are at most $k + 1$.*

Proof:

1. Consider, for instance, complete graphs.
2. On one hand, complete graphs have a twin-cover of 0 but may have arbitrary tree-width. On the other hand, long paths have a tree-width of 1 but may have arbitrary twin-cover.
3. We describe a way of constructing a graph G with twin-cover X by a k-expression. First, we create all vertices in X, each with a unique label, and add edges between them as necessary. Next, to add any non-cover clique in G, we first construct it using label $|X|+1$ (this also requires the use of label $|X| + 2$), then relabel all of its vertices to $|X| + 1$ and then add all necessary edges between the clique and the cover. Once all edges are added, we "forget" its labels (i.e. relabel it to the unused label $|X| + 3$) and proceed to the next clique.
4. We describe a linear rank-decomposition of width $|X|+1$ for any graph G with twin-cover X. Since a linear rank-decomposition is a caterpillar of degree 3, we may characterize it by the sequence of vertices which appear in its nodes of degree 1: first, it contains all vertices in X, followed by all vertices of each non-cover clique in any order of cliques.

   To see that the cut-rank may never exceed $|X| + 1$, consider the following. If the cut splits two vertices in X, then one side of the decomposition only contains at most $|X|$ vertices. Otherwise, the bipartite adjacency matrix will only contain ones in the submatrix between X and non-cover vertices (which has a rank of $|X|$) and a block of ones between a clique we may have cut through. But since we only cut through at most a single clique (due to adding them in sequence), this may only increase the rank by 1.

□

Let us conclude this section with an efficient algorithm to compute a twin-cover when its size is bounded. The fact that this can be done quickly is very important, since otherwise we would need to rely on an oracle or an approximation algorithm to provide the twin-covers before running our parameterized algorithms (as is the case with clique-width [FRRS09]).

Theorem 4 *If a minimum twin-cover in G has size at most k, then it is possible to find a twin-cover of size k in time $O(|E| + k|V| + 1.2738^k)$.*

Proof: To compute the twin-cover, we find and remove all edges between true twins and then compute a minimum vertex cover of the remaining graph (cf. Lemma 2). This second step can be done in time $O(1.2738^k + k|V|)$ [CKX10].

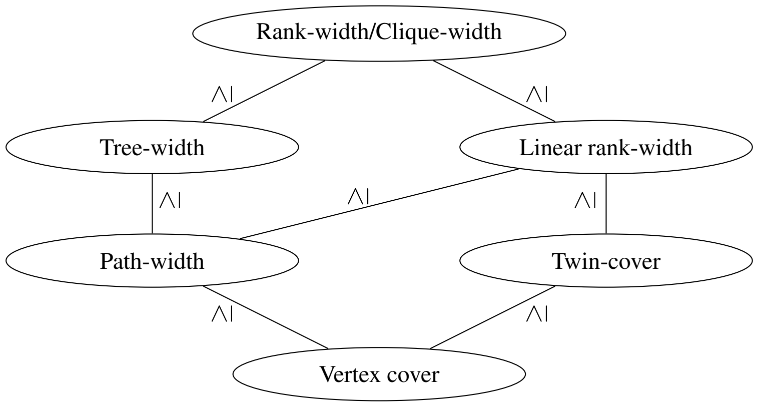

Fig. 3: Relationships between selected graph parameters ($A \geq B$ means that there exists a function f such that for all graphs, $f(A(G)) \geq B(G)$).

Detecting all twin edges is trivial if we allow the algorithm to run through all edges and check whether their incident vertices have the same neighbors, but this could take up to $|E| \cdot |V|$ time. On the other hand, it is possible to improve the running time of this to $O(|E| + |V|)$ by using the following Lemma 5. $\square$

Before we move on to Lemma 5, we need to briefly introduce the area of modular decompositions. Please consult the survey [HP10] or [TCHP08] for a full discussion of this important area.

A *module* S is a set of vertices sharing the same neighbors outside of S – in other words, each vertex $x \notin S$ is either adjacent to every vertex in S, or not adjacent to any vertex in S. A *maximal strong module* is an inclusion-maximal module which does not overlap other modules.

The *modular decomposition tree* (or MD tree) can be recursively defined as follows: the root of the tree corresponds to the entire graph; if the graph is disconnected then the root is called parallel and its children are the MD trees of its components; if the graphs complement is disconnected, the root is called series and its children are the MD trees of the co-components (i.e. the connected components of the complement); in all other cases the root is called prime, and its children are the MD trees of the graphs *maximal strong modules*. Note that each node of the MD tree corresponds to a strong module. The key result of [TCHP08] is that the MD tree can be computed in time $O(|E| + |V|)$.

Lemma 5 *There exists an $O(|E| + |V|)$ algorithm which finds all the twin edges in a graph G.*

Proof: We begin by computing the MD tree. We then look at its root r, and iterate the following procedure $FindTwin(r)$:

- if r is parallel or prime, we run $FindTwin$ on all the children of r;
- if r is series, we add all the vertices which are isolated vertices in the complement of r into a local variable Z. We then mark all edges between vertices in Z as twin edges (it is easy to see that Z forms a clique in G), and run $FindTwin$ (each time with a new, empty Z) on all the children of r which were not added into Z.

For the runtime bound, it suffices to realize that this procedure merely runs through the modular decomposition and performs at most a total of $O(|E|+|V|)$ additional operations.

For correctness, consider any pair of distinct vertices c, d in Z. Any such c, d form true twins, since they are adjacent to each other and to all other vertices in this series node, and also have the same neighborhood outside of this series node.

On the other hand, consider any twin edge ab and let x be the node of the MD tree containing both a and b which has maximum distance from r in the MD tree. This x cannot be a parallel node, since a, b are in the same connected component. Furthermore, if x were a prime node, then a, b would need to be in different maximal strong modules of x, say A, B. But that would imply that $a \cup B$ and $A \cup b$ are also modules of x (since a and b have the same neighborhood), contradicting either the strongness or the maximality of A and B.

So, x is a series node. By our choice of x, it holds that a, b must be in different connected co-components of x. But this means that every other vertex in x is adjacent to both a and b, and so a and b form isolated vertices in the complement of x. $\square$

4 FPT algorithms for problems hard on tree-width

In this section we focus on graph problems where FPT algorithms parameterized by tree-width are not expected to exist. While none of these difficult problems are known to admit a polynomial kernel when parameterized by vertex cover, it is at least possible to obtain FPT algorithms on such graphs. In the following subsections, we provide single-exponential FPT algorithms for BOXICITY, GRAPH MOTIF, EQUITABLE COLORING, PRECOLORING EXTENSION and FIREFIGHTING parameterized by twin-cover.

We remark that all our algorithms implicitly begin by computing a minimum twin-cover (cf. Theorem 4), and that this is included in the presented running times. No lower bounds for any of the following problems are known.

4.1 BOXICITY

The first problem in this section showcases how algorithms parameterized by vertex cover can be trivially generalized to twin-cover.

Definition 5 ([ACS10]) *A k-box is a Cartesian product of closed intervals $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_k, b_k]$. A k-box representation of a graph G is a mapping of the vertices of G to k-boxes in the k-dimensional Euclidean space such that two vertices in G are adjacent iff their corresponding k-boxes have a non-empty intersection. The* boxicity *of a graph G is the minimum integer k such that G has a k-box representation.*

Alternatively, boxicity may be defined as the minimum positive integer b such that the graph can be represented as the intersection of b interval graphs. The notion was first introduced by Roberts [Rob69] in 1969 and has found applications in social sciences and biology. Determining whether the boxicity of a given graph is at most 2 is already NP-hard [Kra94], and it is believed that the BOXICITY problem (which is to compute the boxicity) of graphs of bounded tree-width is NP-hard, however the problem remains open [ACS10].

Adiga, Chitnis and Saurabh [ACS10] introduced two FPT algorithms for BOXICITY parameterized by vertex cover: an exact algorithm with a double-exponential dependency on the parameter, and an algorithm with a single-exponential dependency on the parameter which has a (one-sided) additive error of one. We use the following lemma to extend these results to twin-cover.

Lemma 6 *For any graph G and any pair of true twins $a, b \in V(G)$, deleting b and all of its incident edges does not change the boxicity of G.*

Proof: Consider a minimal k-box representation of a graph G' which was obtained by removing b from G. We may simply add b and all of its incident edges back into G', and map b to a k-box identical to the k-box of a in G'. Thus the boxicity of G is equal to the boxicity of G'. □

Theorem 7

1. *The* BOXICITY *problem can be solved in time $O(|E|^2) + 2^{O(2^k k^2)}|V|$ on graphs of twin-cover at most k.*
2. *The* BOXICITY *problem can be solved with an additive error of one in time $O(|E|^2)+2^{O(k^2 \log k)}|V|$ on graphs of twin-cover at most k.*

Proof: First, we sequentially find a pair of true twins $a, b \in V(G)$ and delete b; by Lemma 6, this does not change the boxicity of G. This procedure takes time at most $O(|E|^2)$. Afterwards the graph contains no more true twins, and hence has vertex cover at most k. At this point it suffices to use the algorithm from either Theorem 1 or Theorem 2 of [ACS10]. □

4.2 GRAPH MOTIF

Definition 6 (GRAPH MOTIF)
Input: A vertex-colored(i) *undirected graph G and a multiset M of colors.*
Question: Does there exist a connected subgraph H of G (called a motif*) such that the multiset of colors $col(H)$ occurring in H is identical to M?*

The GRAPH MOTIF problem was introduced in [LFS06] and arises naturally in bioinformatics, especially in the context of metabolic network analysis. Its complexity has been studied in [FFHV11] and [ABH$^+$10], proving that the problem remains NP-hard even on trees, graphs of path-width 2, superstar graphs and other very restricted classes of graphs.

This subsection will show how to solve GRAPH MOTIF in FPT time parameterized by twin-cover. This is also the first parameterized algorithm for the problem which only parameterizes the structure of the graph and not M.

We will need the following Lemma:

Lemma 8 *Given a* GRAPH MOTIF *instance (G, M), a twin-cover of G and a solution H, either H does not intersect with any vertex in the twin-cover or there exists a solution H' on G', where G' is the subgraph of G obtained by deleting all twin (i.e. uncovered) edges.*

(i) This coloring need not be proper – neighboring vertices may have the same color.

Proof: Assume H intersects the twin-cover. If each non-cover vertex in H is adjacent to a cover vertex in H, then we are done since we can create H' by simply using the edges from the cover vertices of H, which were not deleted.

So, assume for a contradicton that there exists a vertex v in H which is not a cover vertex nor a neighbor of a cover vertex in H. Since H is connected, it contains at least one cover vertex x such that there is a path of non-cover vertices between x and v in H. Then all vertices on this path must be twins, and since one of these vertices is adjacent to x, they all are. So, v had to be adjacent to x. $\square$

This allows us to reduce the problem from a graph of twin-cover k to a graph of vertex cover k. Since GRAPH MOTIF has only been introduced fairly recently, no FPT algorithm parameterized by vertex cover for the problem was known, and so we provide one below.

Theorem 9 *The* GRAPH MOTIF *problem can be solved in time $O(k|V| + 2^{k+k^2} \cdot (|V| + \sqrt{|V|}|E|))$ on graphs of vertex cover number at most k.*

Proof: We assume that the input graph is connected – otherwise the problem may be solved separately for each connected component.

We begin by finding a vertex cover C in time $O(1.2738^k + k|V|)$ [CKX10]. Recall that for any non-cover vertex v, its type $t(v) \subseteq C$ corresponds to its neighborhood in G. Also recall that there are 2^k possible types and that all non-cover vertices are partitioned into sets containing all vertices of the same type.

Next, we run over all possible 2^k subsets C' of C and try to find a motif H which intersects with C exactly in C'. If this is done correctly for all C' and no admissible H is found, it is clear H does not exist. What remains is to decide whether there exists a motif H for a given C'. This is easy if C' is already connected, since we may simply add any adjacent non-cover vertex to H if its color is in $M - col(H)$ (non-existence of a vertex with this color indicates that no H exists for given C').

On the other hand, if C' is not connected, then it is necessary to add up to $k-1$ non-cover vertices to H before it becomes connected—once H becomes connected, then we can once again add pendants greedily as required. One needs to be careful here, since trying $|V|^{k-1}$ possible vertices requires too much time, and, e.g., using some red vertex to connect two components may prevent the use of another red vertex to connect some other components elsewhere (M could contain only a single occurrence of "red"). However, notice that it suffices to select at most one vertex from each type to make H connected. So, we may run over all possible (at most $\left(2^k\right)^k = 2^{k^2}$) k-cardinality sets of types which are used to make H connected. Each type contains vertices with various colors, and we need to make sure to select one color in each type to make H connected so that the occurrence of each color in H does not exceed that in M.

This final subproblem may be solved by finding a maximum matching in the following bipartite graph. The vertex set of this graph contains all colors remaining in the multiset M and one vertex for each type which was pre-selected to be in H. Edges then connect each such type to each color which occurs in that type. A maximum matching in this graph then assigns a color from M to each type. This bipartite graph may be constructed in time $|E|$, has at most $|V|$ vertices and allows a maximum matching to be found in time $O(|V| + \sqrt{|V|}|E|)$ [HK73]. If the resulting maximum matching does not include all type-vertices, then we discard this selection of types.

So, the whole algorithm constructs H by first trying all possible subsets of C in time $O(2^k)$, then trying all possible ways to make H connected by selecting at most $O(2^{k^2})$ possible sets of types, and then deciding which vertices to use from these types based on a maximum matching algorithm in time $O(|V| + \sqrt{|V|}|E|)$ – if the matching does not exist, we skip to the next set of types. At this point we have some "skeleton" of H such that $col(H) \subseteq M$, and we may simply run through all neighbors of H and add everything that is missing from M. This can be done in time $O(|V|)$ and either results in a yes answer or we skip to the next selection of C'. □

Corollary 10 *The* GRAPH MOTIF *problem can be solved in time* $O(k|V| \cdot |M| + 2^{k+k^2} \cdot (|V| + \sqrt{|V|}|E|))$ *on graphs of twin-cover at most* k.

Proof: Given a twin-cover X, we first try whether it is possible to find H in $V - X$; this is easy to do since all that remains are disconnected cliques. Specifically, we compute the multiset of colors contained in each clique and check whether M is a subset of this multiset, with a total time requirement of $|V| \cdot |M|$. Otherwise, we apply Lemma 8 to delete all uncovered edges and obtain a graph with vertex cover number at most k, where we apply Theorem 9. □

4.3 EQUITABLE COLORING

Definition 7 (EQUITABLE COLORING)
Input: A graph $G = (V, E)$ *and a positive integer* r.
Question: Is there a proper vertex coloring using at most r *colors, with the property that the sizes of any two color classes (sets of vertices with the same color) differ by at most one?*

The notion of EQUITABLE COLORING first appeared in 1973 as part of an application for scheduling garbage trucks [Mey73], and has since appeared as a subproblem in various scheduling applications. Fellows et al. proved that it remains W[1]-hard parameterized by tree-width [FFL$^+$11], and Kratochvíl, Fiala and Golovach introduced an FPT algorithm parameterized by vertex cover solving the problem [FGK11], but the presence of large cliques in our graph class requires a new approach to solve the problem.

Theorem 11 *The* EQUITABLE COLORING *problem can be solved in time* $|V||E|r \cdot 2^{O(k \cdot \log k)}$ *on graphs of twin-cover at most* k.

Proof: Notice that for any number of colors c we may trivially compute the number of color classes of size $\lfloor n/c \rfloor$ and of size $\lfloor n/c \rfloor + 1$. Unlike coloring, it is not true that equitable colorability by r colors implies colorability by $r + 1$ colors – for example $K_{n,n}$ is equitably 2-colorable but not 3-colorable. To account for this we run our algorithm for all r possible values of c, and for each choice of c we obtain a fixed (but arbitrarily large) number of color classes with cardinality $\lfloor n/c \rfloor$ and $\lfloor n/c \rfloor + 1$.

The algorithm begins by considering all possible ways of coloring the k cover vertices. However, we do not care about which colors are actually used to color the cover vertices, only whether they are colored by distinct colors and whether that color class has size $\lfloor n/c \rfloor$ or $\lfloor n/c \rfloor + 1$ – specifically, we will account for all possible partitions of the twin-cover into color classes and all possible total sizes of these color classes, without distinguishing between cases which are isomorphic up to color swapping. The number of different partitions of k vertices into (at most k) color sets is upper-bounded by k^k. For each such partition it remains to decide which of the (at most k) color sets containing a cover vertex have size $\lfloor n/c \rfloor$ and which have size $\lfloor n/c \rfloor + 1$. This amounts to at most 2^k possibilities per partition. Note that we discard all precolorings which are not proper.

Now that we have a precoloring of the cover vertices, all that needs to be colored in G are disconnected cliques of various sizes, and all the vertices in each uncolored clique have the same neighbors in the cover. This means that for each clique, we know exactly which colors may or may not be used in that clique, and that each color may be used there only once. This information allows us to construct a network flow instance to color the remaining vertices in G, as follows.

We have a source which is connected by arcs to c vertices representing colors, and the capacity of these arcs is equal to the number of vertices which still need to be colored by that color – for example $\lfloor n/c \rfloor - 2$ if the class was of size $\lfloor n/c \rfloor$ and 2 cover vertices already have this color. Then we create a vertex to represent every uncolored clique in G, and arcs from these vertices to the sink with capacities equal to the size of the clique. Finally, we add arcs of capacity 1 from color vertices to clique vertices if the color may be used in that clique. After computing the maximum flow, we check whether it is equal to the number of uncolored vertices in G (also equal to the sum of arcs outgoing from the source), and if this is the case we immediately obtain a solution in G. Since the maximum flow is bounded by $|V|$, this flow subproblem can be solved in time $O(|V||E|)$ [FF54].

To recapitulate, once we have computed a twin-cover of G, we run through all $c \leq r$ allowed numbers of colors for G in time r. Then we run through all possible ways of partitioning the cover vertices into color classes of sizes $\lfloor n/c \rfloor$ and $\lfloor n/c \rfloor + 1$, which takes time at most $2^{O(k \log k)}$. Finally, we use network flow to decide whether the remaining vertices can be equitably colored with respect to this selection of c, partitioning of cover vertices, and sizes of color classes in this partitioning; this takes time $O(|V||E|)$, and the running time of the whole algorithm then follows. $\square$

4.4 PRECOLORING EXTENSION

PRECOLORING EXTENSION is a natural problem where we are given a partial proper coloring of a graph G and the task is to extend it into a proper coloring of G with r colors. Similarly to EQUITABLE COLORING, it is W[1]-hard when parameterized by tree-width [FFL+11] and FPT when parameterized by vertex cover [FGK11]. The algorithm of [FGK11] cannot be directly applied to twin-cover due to graphs of bounded twin-cover containing large cliques. Here we use a different approach to obtain a single-exponential FPT algorithm parameterized by twin-cover.

Lemma 12 *Given an instance I of* PRECOLORING EXTENSION *on a graph $G = (V, E)$ with a twin-cover X, there exists an instance I' on a graph $G' = (V', E')$ such that:*

1. *I is a yes-instance if and only if I' is a yes-instance,*
2. *$X \subseteq V'$ is also a twin-cover of G',*
3. *G' has at most one non-cover clique in each type,*
4. *I' may be computed in time $O(|V| + |E|)$.*

Proof: Let α_i be the set of colors precolored in type T_i of G and H_i the largest non-cover clique in T_i. We construct G' by first adding the subgraph of G induced by X. Then, for each type T_i of G, we add a single clique T_i' into G' such that each vertex in T_i' is adjacent to the neighbors of T_i in X; T_i' contains $max(|\alpha_i|, |H_i|)$ vertices and has $|\alpha_i|$ vertices precolored with the colors from α_i. Observe that G' may be constructed in linear time, X forms a twin-cover of G', and each T_i' forms a type in G' w.r.t. X.

To prove point 1, assume we have a solution for I. To obtain a solution for I', we start by using the same coloring of X. Then, for each type T'_i, no conflicts with the existing precoloring of vertices are possible, since the precoloring α_i is also present in T_i. If T'_i is not fully precolored, then there must exist a clique H_i in T_i such that $|H_i| = |T'_i|$, and we may use the coloring of H_i in I for T'_i.

On the other hand, assume we have a solution for I'. To obtain a solution for I, we again start by using the same coloring of X. Now each non-cover clique in type T_i may be properly colored with any of the colors in T'_i. □

Theorem 13 *The* PRECOLORING EXTENSION *problem can be solved in time* $2^{O(k\cdot\log k)} \cdot (\sqrt{|V|}|E| + |V|))$ *on graphs of twin-cover at most k.*

Proof: We begin by constructing G' as per Lemma 12. Next, we run through all possible tuples of $(\mathcal{P}, f)$, where $\mathcal{P}$ is a partition of X and f is a mapping from each $S \in \mathcal{P}$ to $\{new, old, fixed\}$. We will refer to these tuples as *pseudo-colorings*, and their purpose is to fix some (partial) information about the coloring of X: which vertices have the same color, and which of these colors already appear precolored somewhere in the graph.

Notice that there are at most $3^k \cdot k^k$ such pseudo-colorings and that for any coloring of G', there exists a pseudo-coloring which "respects" that coloring. Specifically, for any coloring ψ of X, we may construct a pseudo-coloring as follows:

1. $\mathcal{P}$ is the partition of X into equivalence classes with respect to the equivalence of "having the same color in ψ", and
2. $f(S) = fixed$ if S contains a vertex in X which is precolored, and
3. $f(S) = new$ if S is colored in ψ by a color which does not appear precolored anywhere in G'.
4. $f(S) = old$ if S is colored in ψ by a color which appears precolored in G', but not in X.

While running through each possible pseudo-coloring, we first check whether the pseudo-coloring is valid (i.e. whether the information it captures is relevant for some coloring of X). A pseudo-coloring is valid if:

a) each $S \in \mathcal{P}$ is independent,

b) each new and each old $S \in \mathcal{P}$ only contains unprecolored vertices of X,

c) each fixed $S \in \mathcal{P}$ contains at least one precolored vertex in X and all its precolored vertices have the same color,

d) any two vertices $v, u \in X$ precolored with the same color are contained in the same $S \in \mathcal{P}$.

Next, for each valid pseudo-coloring, we check whether it is possible to assign colors from the precoloring to the old $S \in \mathcal{P}$ without creating conflicts. This may be done by finding a maximum matching between the old S and the precolored colors: a precolored color a is adjacent to an old $S \in \mathcal{P}$ iff no vertex in s is adjacent to a vertex precolored with a in G'. Such a bipartite graph may be constructed in time $|E|$, has at most $|V|$ vertices and allows a maximum matching to be found in time $O(|V| + \sqrt{|V|}|E|)$ [HK73].

If the resulting maximum matching does not include all old S, we discard our pseudo-coloring and move on to the next one; otherwise we use the matching to designate specific colors to each individual old S.

Each (undiscarded) pseudo-coloring is then greedily extended to a full coloring of G'. For each non-cover clique H' in G', we add all colors which already appear in G' but are not adjacent to H'. If there remain uncolored vertices in H', we use a new color for each. For each pseudo-coloring, we then compare the total number of colors used to the r specified in the input. All of this takes time at most $O(|E|+|V|)$.

For correctness, assume that there exists a precoloring extension χ of G' with r colors. Then there exists a pseudo-coloring $(\mathcal{P}, f)$ which respects this χ, and at least one matching between all the old $S \in \mathcal{P}$ and the precolored colors in G'. Notice that, for any $Z \subseteq X$, the number of distinct $S \in \mathcal{P}$ which intersect with Z is equal to the number of the distinct colors of Z in χ, and so the number of colors in any such Z is the same. Let Z_i be the neighborhood of each type T_i in X, and let c_i be the number of colors used to color Z_i. Regardless of which specific colors are used for Z_i in the solution found by our algorithm, there are no conflicts between Z_i and T_i (due to the matching subroutine). Furthermore, $c_i + |T_i| \leq r$, since T_i may be colored by at most r colors in χ. But this implies that the coloring obtained greedily by our algorithm for this choice of $(\mathcal{P}, f)$ will never use more than r colors for each T_i. $\square$

4.5 FIREFIGHTING

Definition 8 (Firefighting, [FKMR07])
Input: A graph $G = (V, E)$ and a vertex $r \in V(G)$.
At time 0, a fire breaks out at r. At each subsequent time interval, the firefighter defends some vertex which is not on fire, and then the fire spreads to all undefended neighbors of each vertex which is on fire. Once defended, a vertex remains so for all time intervals. The process ends when the fire can no longer spread.
Question: What is the maximum number of vertices which can be saved, i.e., that are not burning when the process ends?

The FIREFIGHTING problem was first introduced by Hartnell [Har95] in 1995 and can be used to model the spread of fire, diseases or computer viruses. It was shown to be NP-complete even for trees of maximum degree 3 by Finbow et al. [FKMR07]. The parameterized complexity of the problem has been studied in several papers, for instance [BCF11, CFvL11], with a primary focus on parameterizing by the number of vertex saved (or burned) in the decision version of the problem.

While an FPT algorithm parameterized by the vertex cover together with the number of vertices which may burn was published in [BCF11], a purely structural parameterization of the optimization variant has not yet been considered. We approach the problem as follows: first, we use an observation to reduce an instance of bounded twin-cover to bounded vertex cover, and then design an FPT algorithm for the FIREFIGHTING problem parameterized by vertex cover.

Observation 14 Assume the fire starts on a cover vertex. Then the fire will never spread through uncovered edges: whenever one endpoint of an uncovered edge catches on fire, the other endpoint also catches on fire simultaneously (since they are true twins).

On the other hand, assume the fire starts on a non-cover vertex. If the vertex is only in an isolated clique, then the (obvious) best strategy is to defend at least a single vertex in the clique before the fire spreads. If the vertex is in a clique adjacent to a single cover vertex, the optimal strategy is to defend that cover vertex.

Finally, what happens if the fire starts on a non-cover vertex connected to at least two cover vertices? In two turns all of the undefended vertices of that type will be on fire, regardless of how they are partitioned into cliques (first the fire spreads to at least one cover vertex, and from there to the whole type).

Now all that remains is to solve the problem. We will first need a simple lemma:

Lemma 15 *Given an instance of the firefighting problem on G, the fire can only spread at most $2k + 1$ times, k being the twin-cover of G.*

Proof: Each time the fire spreads, it moves along an induced path in G. The maximum length of an induced path in G is $2k + 1$. □

Theorem 16 *The* FIREFIGHTING *problem can be solved in time $O(|E| + (2^k + k)^{2k+1} \cdot |V|)$ on graphs of twin-cover at most k.*

Proof: Observation 14 allows us to delete all uncovered edges and work with a graph of vertex cover at most k (unless the vertex on fire is in an isolated clique or a clique with a single adjacent cover vertex – these may be easily solved separately). Additionally, since all the vertices in each type are symmetrical to each other, it never matters which vertex we defend within a given type.

Each time we need to choose a vertex to defend, we only consider at most 2^k types + k cover vertices to choose from. Additionally, we only have to defend at most $2k+1$ times thanks to Lemma 15. Altogether, we have a total of at most $(2^k + k)^{2k+1}$ possible "firefighting strategies" (sequences of vertices to defend), and to obtain the solution it suffices to try each "strategy" and count the number of vertices on fire in time $O(|V|)$. □

5 Other FPT and kernelization algorithms

5.1 FPT algorithms for problems hard on clique-width

Here we present two FPT algorithms for problems which are not in FPT when parameterized by clique-width: CHROMATIC NUMBER and MAX-CUT. While these two problems are known to be in FPT parameterized by tree-width, our results may be used to solve them efficiently even on well-structured dense graphs.

5.1.1 CHROMATIC NUMBER

The well-known CHROMATIC NUMBER problem is to compute the minimum number of colors required to color the vertices of a graph so that no pair of adjacent vertices have the same color (such a coloring is called *proper*). It has been shown that, although being FPT, the problem is very unlikely to admit a polynomial kernel when parameterized by vertex cover [BJK11]. Furthermore, it is not in FPT when parameterized by clique-width [FGLS09] assuming the Exponential Time Hypothesis holds. While this rules out the existence of a polynomial kernel parameterized by twin-cover, we at least prove that this parameterized problem is FPT.

Theorem 17 CHROMATIC NUMBER *can be solved in time $O(k|V| + k^k|E|)$ on graphs of twin-cover at most k.*

Proof: First, we run through the at most k^k ways of precoloring the twin-cover with colors $1 \dots k$ and check if each such coloring is proper. If our precoloring is proper, we find a minimum coloring of the remaining vertices.

This may be done as follows: We run through all non-cover cliques in G, and for each such clique of size i we read the set of colors P appearing in its neighborhood (in the twin-cover). Then we color the clique with i colors Q such that $Q \cup P = \{1, 2 \dots i + |P|\}$ and $Q \cap P = \emptyset$ (in other words, we color the clique using as few new colors as possible without using colors from its neighborhood). All of this may be done in time at most $O(|E|)$ for each precoloring of the twin-cover, and for each such precoloring we store the total number of colors used. In the end, we output the minimum number of total colors used for some proper precoloring of the twin-cover.

For correctness, assume that there exists a better solution than the one found by the algorithm. Then the coloring of the twin-cover in this optimal solution is the same (up to renaming of colors) as some precoloring found in our algorithm. The optimal solution would then have to use less colors to color some non-cover clique, but that is clearly not possible. □

5.1.2 MAX-CUT

Definition 9 (MAX-CUT)
Input: A graph $G = (V, E)$ and a number $c \in \mathbb{N}$.
Question: Does there exist a set $S \subseteq V$ such that the number of edges between S and $V - S$ is greater or equal to c?

The MAX-CUT problem is known to be FPT parameterized by tree-width [Wim87], but is W[1]-hard when parameterized by clique-width [FGLS10]. It is not known whether the problem admits a polynomial kernel when parameterized by vertex cover. Here we provide a simple FPT algorithm parameterized by twin-cover:

Theorem 18 MAX-CUT *can be solved in time $O(2^k \cdot |V||E|)$ on graphs of twin-cover at most k.*

Proof: First, we run through all 2^k possible intersections of S with the twin-cover X. For each such $S \cap X$, we may calculate the optimal intersections of S with each non-cover clique separately, since all non-cover cliques are independent. So for each non-cover clique Z, we try $|Z| + 1$ possible ways of cutting it by running through $0 \leq |S \cap Z| \leq |Z|$ and for each we compute the value of the cut between $S \cap (Z \cup X)$ and $(Z \cup X) - S$ in time $|E|$. For each non-cover clique we then fix $S \cap Z$ to the value which maximized the cut.

Altogether this gives an $O(2^k \cdot |V||E|)$ FPT algorithm, since for each fixed $S \cap X$ we run through at most $|V|$ possibilities for various $S \cap Z$. □

5.2 Using twin-cover for kernelization

Vertex cover has also been studied extensively as a parameter in the context of kernelization. For instance, both the HAMILTONIAN CYCLE and VERTEX-DISJOINT PATHS problems admit polynomial kernels parameterized by vertex cover. Similarly to the results concerning FPT algorithms, problems which admit polynomial kernels when parameterized by vertex cover also often admit such kernels when parameterized by twin-cover.

Bodlaender, Jansen and Kratsch [BJK11] proved that HAMILTONIAN CYCLE and VERTEX-DISJOINT PATHS as well as other problems admit polynomial kernels when parameterized by a more general parameter called *cluster vertex deletion distance* – the vertex deletion distance to a disjoint union of cliques. Polynomial kernels for these problems parameterized by twin-cover immediately follow from their results. On the other hand, there are problems of interest which are in FPT parameterized by twin-cover but are W[1]-hard when parameterized by cluster vertex deletion distance (for instance PRECOLORING EXTENSION [DK12]).

In light of these facts, we only illustrate the basic kernelization techniques for twin-cover on the simple example below.

5.2.1 FEEDBACK VERTEX SET

Definition 10 (FEEDBACK VERTEX SET)
Input: A graph $G = (V, E)$ and a number $u \in \mathbb{N}$.
Question: Does there exist a set $S \subseteq V, |S| \leq u$ such that $G - S$ is acyclic?

Bui-Xuan, Telle and Vatshelle showed that the FEEDBACK VERTEX SET problem is known to be FPT parameterized by clique-width [BXTV09] and Thomassé proved that it admits a quadratic kernel when parameterized by $|S|$ [Tho09].

Proposition 19 *It is possible to compute a kernel with at most $5k^2 + k$ vertices for* FEEDBACK VERTEX SET *in polynomial time on a graph G when a twin-cover X of size at most k is provided.*

Proof: Given a graph G with a twin-cover X, we run through each non-cover clique Z such that $|Z| \geq 3$. For each, we delete $|Z|-2$ vertices and reduce u by $|Z|-2$. If u becomes negative, we have a no-instance.

Thus we get a subgraph $G' \subseteq G$ with the same twin-cover X as G such that $G' - X$ is acyclic. Then the feedback vertex set is at most k, and we may use the result of [Tho09] to get a $5k^2 + k$ kernel in polynomial time. □

6 Monadic Second-Order Logic

Monadic second-order logic is used in algorithmic graph theory to describe a wide range of problems on graphs. We distinguish between the so-called MS_2 logic, which allows quantification over (sets of) vertices and edges, and the weaker MS_1 logic, which only allows quantification over (sets of) vertices.

Definition 11 (MS_1 logic of graphs) *The language of MS_1 consists of expressions built from the following elements:*

- *variables $x, y, \ldots$ for vertices, and $X, Y, \ldots$ for sets of vertices,*
- *the predicates $x \in X$ and $edge(x, y)$ with the standard meaning,*
- *equality for variables, quantifiers $\forall, \exists$ ranging over vertices and vertex sets, and the standard Boolean connectives.*

It is well known that all MS_2-definable problems can be solved in FPT time when parameterized by tree-width, and the same is true for MS_1-definable problems when parameterized by rank-width. Unfortunately, in both of these cases there is a non-elementary dependence between the running time and the formula – specifically, the height of the tower of exponents grows with the number of quantifier alternations in the formula. Even worse, Frick and Grohe proved that this non-elementary dependence on the parameter is unavoidable (unless P=NP) [FG04]. This means that while complicated formulas are still solvable in FPT time on these graph classes, the time complexity would be enormous in practice.

The purpose of this section is to show that MS_1 model checking can be done in elementary FPT time parameterized by twin-cover. We use the recent results of [GHN$^+$12] to obtain a straightforward proof of this fact. The second part of the section discusses the possibility of improving the running time of MS_1 model checking up to the lower bounds proved by Lampis in [Lam12].

6.1 MS_1 model checking parameterized by twin-cover

One of the results of [GHN$^+$12] is that MS_1 model checking can be done in elementary FPT time when parameterized by the so-called *shrub-depth*. Here the role of shrub-depth is only to ensure that, unlike in the case of tree-width or clique-width, the *tree-model* of the graphs has bounded depth. In this subsection, we prove that graphs of bounded twin-cover also have bounded shrub-depth. The advantage of this approach is that it allows a simple direct proof of Corollary 22, without the need to add multiple pages of new definitions, formalisms and case analysis.

We proceed as follows. First, we introduce the notion of *tree-models*, which are then used to define shrub-depth. Then, we establish a connection between shrub-depth and twin-cover in Theorem 21.

Definition 12 (Tree-model) *We say that a graph G has a* tree-model of m labels and depth d *if there exists a rooted tree T such that*

i) the set of leaves of T is exactly $V(G)$,

ii) the length of each root-to-leaf path in T is exactly d,

*iii) each leaf of T is assigned one of m labels (T is m-*labelled*),*

iv) and the existence of a G-edge between $u, v \in V(G)$ depends solely on the labels of u, v and the distance between u, v in T.

The class of all graphs having a tree-model of m labels and depth d is denoted by $\mathcal{TM}_m(d)$.

Definition 13 (Shrub-depth) *A class of graphs $\mathcal{G}$ has* shrub-depth d *if there exists m such that $\mathcal{G} \subseteq \mathcal{TM}_m(d)$, while for all natural m it is $\mathcal{G} \not\subseteq \mathcal{TM}_m(d-1)$.*

Corollary 20 (Corollary 4.4 of [GHN$^+$12]) *Assume $d \geq 1$ is a fixed integer. Let $\mathcal{G}$ be any graph class of shrub-depth $\leq d$. Then the MS_1 model checking problem on $\mathcal{G}$, i.e., testing $G \models \phi$ for the input $G \in \mathcal{G}$ and MS_1 sentence ϕ, can be solved by an FPT algorithm, the runtime of which has an elementary dependence on the parameter and ϕ. This assumes G is given on the input alongside with its tree-model of depth d.*

Theorem 21 *Let $\mathcal{H}_k$ be the class of graphs of twin-cover k. Then $\mathcal{H}_k \subseteq \mathcal{TM}_{2^k+k}(2)$ and a tree-model of any $G \in \mathcal{H}_k$ may be constructed in single-exponential FPT time.*

Proof: For any $G \in \mathcal{H}_k$ with twin-cover X, we construct its tree model as follows: At distance 2 from the root, we label each vertex $v \in X$ by a separate label $\beta_1 \dots \beta_k$ and each vertex $v \notin X$ by label $i : v \in T_i$, where $T_1, T_2 \dots T_{2^k}$ are the types of X. At distance 1 from the root, we merge leaves u, v into a single branch iff u, v are in the same non-cover clique in G.

What remains is to check condition iv) of Definition 12. Since each vertex in X has a separate label, all edges induced by X satisfy condition iv). For edges between X and $V - X$, any vertices $a, b \notin X$ in the tree-model with the same label i have the same neighborhood in X. Finally, for edges between $u, v \notin X$, we know that such u, v are true twins and so are at distance 2 in the tree-model.

On the other hand, observe that for any pair of vertices $u, v \in G$ which are not adjacent, there cannot exist a vertex u' with the same label as u and at the same distance to v as u such that $u'v \in E$. □

Then from Corollary 20 and Theorem 21 we get:

Corollary 22 *The MS_1 model checking problem on the class of graphs of twin-cover at most k can be solved by an FPT algorithm with an elementary dependence on k and the formula.*

We conclude this section with a note on the running time of Corollary 22. While the runtime of Corollary 20 is not mentioned explicitly in [GHN$^+$12], the application of Theorem 21 onto Theorem 4.3 of [GHN$^+$12] (which is used to obtain Corollary 20) results in a runtime which is quadruple-exponential in k and triple-exponential in $|\phi|$.

7 Concluding notes

We believe that the presented algorithmic results provide more than enough evidence that twin-cover is a useful and powerful parameter. Potential applications of twin-cover include:

1. Replacing vertex cover in the design of FPT algorithms for various problems which are W[1]-hard when parameterized by tree-width (or even NP-hard for constant values of tree-width).
2. Providing a more viable alternative to tree-width on dense graph classes for problems which are W[1]-hard when parameterized by clique-width (or even NP-hard for constant values of clique-width).
3. Obtaining polynomial kernels on richer graph classes.
4. Faster MS_1 model checking on certain graph classes.

Future work should focus on the practical aspects of twin-cover. How much smaller is the twin-cover of various graphs compared to their vertex cover, and how does it compare to tree-width? How much of a speed-up does it actually provide over vertex cover? One would expect the runtime differences to be huge, however the actual numbers are yet to be seen.

As for limitations and open questions, there exist MS_2-definable problems which cannot be solved in polynomial time even on complete graphs unless $EXP = NEXP$ [CMR00]. Such a problem admits a single-exponential FPT algorithm parameterized by vertex cover, but is not in FPT when parameterized by twin-cover. That being said, the vast majority of graph problems are trivial on complete graphs. Perhaps the most interesting set of problems parameterized by twin-cover which remain open are the graph layout problems studied, e.g., in [FGK11], which are in FPT when parameterized by vertex cover.

Finally, an inquisitive reader might be wondering whether we could generalize our notion even further. For instance, it would be tempting to replace a bounded number of cover vertices by a bounded number of “cover cliques”, where an edge would be covered if it were incident to some vertex in a cover clique. The problem with this, as well as several other, approaches is that such a parameter would be low on split graphs (graphs which may be partitioned into a clique and an independent set). Several problems of interest are already known to be NP-hard on split graphs. As a final result, we prove that GRAPH MOTIF is one such problem.

Theorem 23 *The* GRAPH MOTIF *problem is NP-hard on split graphs.*

Proof: Our proof is based on a polynomial reduction from 3SAT. For any 3SAT instance consisting of variables $VAR = \{x_1 \ldots x_j\}$ and clauses C, let M contain precisely one unique color for each variable and one unique color for each clause. G will consist of a clique K, $|K| = 2 \cdot |VAR|$, with every variable color occurring exactly twice in this clique. So, every variable corresponds to two vertices in K, one representing a truth assignment and the other representing false. Finally, we add an independent set C with each vertex colored by a unique clause color, and connect each vertex by edges to the three vertices representing the three literals occurring in that clause.

It is easy to see that a yes instance of GRAPH MOTIF on this graph is equivalent to a solution to the original 3SAT instance. Such an H contains exactly one vertex in K for each variable (which vertex is chosen determines whether that variable should be true or false in the solution to the SAT instance), and every clause vertex contains an edge to some selected vertex, which translates to the clause being satisfied by that variable in the 3SAT instance. On the other hand, any satisfying assignment of variables in 3SAT immediately provides a solution to the GRAPH MOTIF instance, since it tells us which vertices to choose in K so that it is possible to reach all the clause vertices. □

References

[ABH$^+$10] Abhimanyu M Ambalath, Radheshyam Balasundaram, Chintan Rao H, Venkata Koppula, Neeldhara Misra, Geevarghese Philip, and M S Ramanujan. On the kernelization complexity of colorful motifs. In *IPEC 2010*, number 6478 in LNCS, pages 14–25, 2010.

[ACS10] Abhijin Adiga, Rajesh Chitnis, and Saket Saurabh. Parameterized algorithms for boxicity. In *ISAAC (1)*, pages 366–377, 2010.

[BCF11] Cristina Bazgan, Morgan Chopin, and Michael R. Fellows. Parameterized complexity of the firefighter problem. In *ISAAC*, pages 643–652, 2011.

[BDFH09] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[BJK11] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *STACS*, pages 165–176, 2011.

[Bod09] Hans L. Bodlaender. Kernelization: New upper and lower bound techniques. In *IWPEC*, pages 17–37, 2009.

[BXTV09] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low cliquewidth. In *IWOCA*, pages 113–124, 2009.

[CFvL11] Marek Cygan, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting revisited. In *IPEC*, pages 13–26, 2011.

[CKX10] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411:3736–3756, 2010.

[CMR00] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer, 1999.

[Die05] R. Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, New York, 2005.

[DK12] Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *MFCS*, pages 348–359, 2012.

[EFG$^+$09] Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad Kanj, Frances Rosamond, and Ondřej Suchý. What makes equitable connected partition easy. In *IPEC*, pages 122–133, 2009.

[FF54] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1954.

[FFHV11] Michael R. Fellows, Guillaume Fertin, Danny Hermelin, and Stéphane Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.*, 77:799–811, July 2011.

[FFL$^+$11] Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209:143–153, February 2011.

[FG04] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[FGK11] Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011.

[FGLS09] F. Fomin, P. Golovach, D. Lokshtanov, and S. Saurab. Clique-width: On the price of generality. In *SODA'09*, pages 825–834. SIAM, 2009.

[FGLS10] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized with clique-width. In *SODA*, pages 493–502, 2010.

[FHR12] Michael R. Fellows, Danny Hermelin, and Frances A. Rosamond. Well quasi orders in subclasses of bounded treewidth graphs and their algorithmic applications. *Algorithmica*, 64(1):3–18, 2012.

[FKMR07] Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discrete Mathematics*, 307(16):2094–2105, 2007.

[FLM$^+$08] Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC (1)*, pages 294–305, 2008.

[FRRS09] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discret. Math.*, 23:909–939, May 2009.

[Gan10] Robert Ganian. Thread graphs, linear rank-width and their algorithmic applications. In *IWOCA*, pages 38–42, 2010.

[Gan11] Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC*, number 7112 in LNCS, pages 259–271, 2011.

[GH10] Robert Ganian and Petr Hliněný. On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010.

[GHN$^+$12] Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *MFCS*, pages 419–430, 2012.

[Har95] B. Hartnell. Firefighter! An application of domination. Presentation. In *25th Manitoba Conference on Combinatorial Mathematics and Computing*, University of Manitoba in Winnipeg, Canada, 1995.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[HP10] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[Kra94] Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Appl. Math.*, 52:233–252, August 1994.

[Lam12] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

[LFS06] Vincent Lacroix, Cristina G. Fernandes, and Marie-France F. Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 3(4):360–368, 2006.

[Mey73] W. Meyer. Equitable coloring. *American Mathematical Monthly*, 80:920–922, 1973.

[OS06] S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[Rob69] F. S. Roberts. On the boxicity and cubicity of a graph. In *Recent Progresses in Combinatorics*, pages 301–310. Academic Press, 1969.

[RS90] N. Robertson and P. D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Combin. Theory Ser. B*, 48(3):227–254, 1990.

[TCHP08] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP (1)*, pages 634–645, 2008.

[Tho09] Stéphan Thomassé. A quadratic kernel for feedback vertex set. In *SODA '09*, pages 115–119, 2009.

[Wim87] Thomas Victor Wimer. *Linear algorithms on k-terminal graphs*. PhD thesis, Clemson University, 1987.