Algorithms and Data Structures for First-Order Logic with Connectivity Under Vertex Failures

Michał Pilipczuk
University of Warsaw, Poland

Nicole Schirrmacher
Universität Bremen, Germany

Sebastian Siebertz
Universität Bremen, Germany

Szymon Toruńczyk
University of Warsaw, Poland

Alexandre Vigny
Universität Bremen, Germany

Abstract

We introduce a new data structure for answering connectivity queries in undirected graphs subject to batched vertex failures. Precisely, given any graph G and integer parameter k, we can in fixed-parameter time construct a data structure that can later be used to answer queries of the form: "are vertices s and t connected via a path that avoids vertices $u_1, \ldots, u_k$?" in time $2^{\mathcal{O}(k)}$. In the terminology of the literature on data structures, this gives the first deterministic data structure for connectivity under vertex failures where for every fixed number of failures, all operations can be performed in constant time.

With the aim to understand the power and the limitations of our new techniques, we prove an algorithmic meta theorem for the recently introduced *separator logic*, which extends first-order logic with atoms for connectivity under vertex failures. We prove that the model-checking problem for separator logic is fixed-parameter tractable on every class of graphs that exclude a fixed topological minor. We also show a weak converse. This implies that from the point of view of parameterized complexity, under standard complexity theoretical assumptions, the frontier of tractability of separator logic is almost exactly delimited by classes excluding a fixed topological minor.

The backbone of our proof relies on a decomposition theorem of Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [SICOMP '19], which provides a tree decomposition of a given graph into bags that are unbreakable. Crucially, unbreakability allows to reduce separator logic to plain first-order logic within each bag individually. Guided by this observation, we design our model-checking algorithm using dynamic programming over the tree decomposition, where the transition at each bag amounts to running a suitable model-checking subprocedure for plain first-order logic. This approach is robust enough to provide also an extension to efficient enumeration of answers to a query expressed in separator logic.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Finite Model Theory; Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorics and graph theory, Computational applications of logic, Data structures, Fixed-parameter algorithms and complexity, Graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.102

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2111.03725

Funding This paper is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 948057) and part of the French-German Collaboration ANR/DFG Project UTMA supported by the German Research Foundation (DFG) through grant agreement No 446200270.

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).
Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;
Article No. 102; pp. 102:1–102:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Ismaël Jecker, Pierre Ohlmann and Wojciech Przybyszewski for useful discussions. In particular, Wojciech Przybyszewski showed how to improve the dependency on k in the running time from double exponential to single exponential in Theorem 1.1 (see Theorem 2.4).

1 Introduction

1.1 Connectivity under vertex failures

In many applications, we do not need to answer a query once but rather need to repeatedly answer queries over dynamically changing data. A prime example are databases, where a database is repeatedly modified and queried by the users. In most cases, the modifications to the database can be expected to be small compared to its size, which suggests that in a preprocessing step, we can set up a data structure that is efficiently updated after each modification and that allows for efficient querying. In 1997, Frigioni and Italiano introduced the *dynamic subgraph model* [22], which allows to conveniently model such dynamic situations in the particular case of connectivity in network infrastructures that are subject to node or link failures. This model, which has been studied intensively in the data structures community under the name of *connectivity oracles under vertex failures*, is as follows. We are given a graph G and an integer k. We imagine that G is a network in which at every moment, at most k vertices (or edges) are inactive (are subject to *failures*). The goal is to construct a data structure that supports the following two operations. First, one can *update* G by resetting the set of failed vertices to a given set of size at most k. Second, one can *query* G by asking, for a given pair of vertices s and t, whether s and t can be connected by a path that avoids the failed vertices.

Following the introduction of the problem by Frigioni and Italiano [22] and the more general problem of batched vertex or edge failures by Pătraşcu and Thorup [37][1], there has been a long line of work on data structures for connectivity oracles under vertex and edge failures. We refer to a remarkably comprehensive literature overview in the work of Duan and Pettie [17], which also provides the currently best bounds for the problem as far as combinatorial and deterministic data structures are concerned. Their data structure can be initialized in time $\mathcal{O}(|G|\|G\| \cdot \log|G|)$, takes $\mathcal{O}(k\|G\| \log|G|)$ space, and supports updates in $\mathcal{O}(k^3 \log^3|G|)$ time and queries in $\mathcal{O}(k)$ time. Here $|G|$ is the vertex count of G and $\|G\|$ is the joint number of edges and vertices in G. These bounds can be slightly improved at the cost of allowing randomization, however, the polylogarithmic dependency on the size of the graph in the update time persists. As noted in [17], from known results it is possible to derive data structures with constant time complexity of operations for $k \leqslant 3$ (or $k \leqslant 4$ for edge failures), but, citing their words, "scaling these solutions up, even to an arbitrarily large constant k, becomes prohibitively complex, even in the simpler case of edge failures."

Recently, van den Brand and Saranurak [42] proposed a very different approach to the problem, using which they obtained a randomized data structure that can be initialized in time $\mathcal{O}(|G|^\omega)$, takes $\mathcal{O}(|G|^2 \log|G|)$ space, and supports updates in $\mathcal{O}(k^\omega)$ time and queries in $\mathcal{O}(k^2)$ time (where ω is the exponent for the boolean matrix multiplication).

In particular, the update and the query time are constant for constant k; to the best of our knowledge, this is the only data structure that has this property known so far. The approach is algebraic and, simplifying it substantially, boils down to storing a matrix of

[1] Strictly speaking, in [37] Pătraşcu and Thorup considered only edge failures. To the best of our knowledge, (batched) vertex failures on general graphs were first investigated by Duan and Pettie [16].

counts of walks between pairs of vertices and extracting answers to queries using algebraic operations on those counts. To manipulate the counts efficiently, one needs to work on their short hashes, for instance in the form of elements from a fixed finite field. This introduces randomization and avoiding it within this methodology seems difficult. We remark that the data structure of van den Brand and Saranurak [42] works even for the more general problem of reachability in directed graphs under vertex failures, and can handle arc insertions.

As the first main contribution of this paper, we prove the following theorem. Note that querying whether two vertices are connected by a path avoiding failed vertices is equivalent to an update followed by a connectivity query in the terminology of previous works [42, 16, 17, 37].

▶ **Theorem 1.1.** *Given a graph G and an integer k, one can in time $2^{2^{\mathcal{O}(k)}} \cdot |G|^2\|G\|$ construct a data structure that may answer the following queries in time $2^{\mathcal{O}(k)}$: given $s,t \in V(G)$ and k vertices $u_1,\ldots,u_k$ of G, are s and t connected in G by a path that avoids the vertices $u_1,\ldots,u_k$? The space usage of the data structure is $2^{2^{\mathcal{O}(k)}} \cdot \|G\|$.*

Note that in Theorem 1.1, for every fixed k, all operations are supported in constant time (which is exponential in k). Also, the data structure is entirely deterministic and purely combinatorial. To the best of our knowledge, this is the first deterministic (non-randomized) data structure with constant time queries. Note also that for a fixed k, the space usage of our data structure is linear in the size of the graph, as opposed to the quadratic dependency in the result of van den Brand and Saranurak [42].

We remark that in the first version of this paper [36], the query time in Theorem 1.1 is stated as $2^{2^{\mathcal{O}(k)}}$ instead of $2^{\mathcal{O}(k)}$. The improved query time is due to an improvement in one of the ingredients of our proof (see Theorem 2.4 below), provided by Wojciech Przybyszewski.

The doubly-exponential dependencies on k may seem high, however we also derive two variants of our data structure that achieve the following tradeoffs:

- The space usage and the query time can be reduced to $2^{\mathcal{O}(k^2)} \cdot \|G\|$ and $k^{\mathcal{O}(1)}$, respectively, while the construction time becomes $2^{\mathcal{O}(k^2)} \cdot |G|^{\mathcal{O}(1)}$.
- The space usage and the query time can be replaced with $k^{\mathcal{O}(1)} \cdot |G|^2$ and $k^{\mathcal{O}(1)}$, respectively, while the construction time becomes $2^{\mathcal{O}(k \log k)} \cdot |G|^{\mathcal{O}(1)}$.

Note the first statement offers both more efficient queries (the query time is even polynomial in k), and smaller space usage. The slight drawback is that the polynomial factor in the construction time becomes unspecified, but it is still of the form $|G|^{\mathcal{O}(1)}$. In the second statement, the space usage becomes polynomial in k at the expense of making it quadratic in $|G|$. The tradeoffs are obtained by a non-trivial replacement of particular components in the proof of Theorem 1.1; see the discussion in Section 6 of the full version. We consider the statement provided in Theorem 1.1 to be the cleanest formulation, hence we put a primary focus on it.

As for the proof of Theorem 1.1, our approach is completely new compared to the previous approaches [42, 16, 17, 37]. Our key combinatorial observation is that connectivity queries over a constant number k of vertex failures can be evaluated in constant time provided the underlying graph G is sufficiently well-connected. The requirement on the well-connectedness of G depends on the number k. Obviously, there is no guarantee that the input graph G satisfies this property. We therefore use a decomposition theorem of Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [15] that (roughly) states the following (see Theorem 2.1). For every graph G and fixed number k, there is a tree decomposition where every intersection of adjacent bags has bounded size and every bag is well-connected for parameter k. Moreover, such a tree decomposition can be computed in time $2^{\mathcal{O}(k^2)}|G|^2\|G\|$. The data structure of

Theorem 1.1 is constructed around the tree decomposition $\mathcal{T}$ provided by the theorem of Cygan et al. [15]. Intuitively speaking, whether s and t are connected via a path that avoids the vertices $u_1, \ldots, u_k$ can be decided using dynamic programming over $\mathcal{T}$. We employ known techniques developed for answering queries on trees in constant time, for instance a deterministic variant of Simon's factorization [12], to be able to compute the outcome of this dynamic programming efficiently for any query given on input.

We provide a more detailed overview of the proof of Theorem 1.1 in Section 2. All proofs are provided in the full version [36] (numbers referring to `https://arxiv.org/abs/2111.03725v1`).

1.2 Algorithmic meta theorems

Once an algorithmic technique has been applied to solve a specific problem, it is very desirable to understand the power and the limitations of that technique, that is, to understand which other problems can be solved with that technique and which problems cannot. An approach to this very general goal is to prove an *algorithmic meta theorem*: a theorem that establishes tractability for a whole class of problems, possibly on certain restricted input instances. As logic allows to conveniently define classes of algorithmic problems, namely, the class of all problems that can be expressed in the logic under consideration, algorithmic meta theorems are often formulated as model-checking problems for a logic. In the *model-checking problem* for a logic $\mathcal{L}$ on a class of graphs $\mathscr{C}$ we are given a graph $G \in \mathscr{C}$ and a sentence $\varphi \in \mathcal{L}$, and the task is to decide whether φ holds in G. The archetypal result of this form is a result of Courcelle stating that every formula of monadic-second order logic (MSO_2) can be evaluated in linear time on all graphs whose treewidth is bounded by some fixed constant [13]. This result has been extended to various other logics $\mathcal{L}$ and graph classes $\mathscr{C}$.

This approach to algorithmic meta theorems can be made precise in the language of parameterized complexity, as follows. Say that $\mathcal{L}$ is *tractable* on $\mathscr{C}$ if the model-checking problem for $\mathscr{C}$ and $\mathcal{L}$ is *fixed-parameter tractable (fpt)*: it can be solved in time $f(\varphi) \cdot \|G\|^c$, for some computable function f and a constant c, both depending only on $\mathscr{C}$. Results about tractability of a logic $\mathcal{L}$ on a class $\mathscr{C}$ imply the tractability of a vast array of problems – namely all problems that can be expressed in the logic $\mathcal{L}$ – on a given class of graphs. For instance, if model-checking *first-order logic* (FO) is fpt on a class of graphs $\mathscr{C}$, then in particular the dominating set problem is fpt on $\mathscr{C}$, since the existence of a dominating set of size k can be expressed by a first-order sentence with $k+1$ quantifiers that range over the vertices of the graph. Similarly, the independent set problem is then also fpt on $\mathscr{C}$. On the other hand, if the more powerful logic MSO_2 is tractable on a class of graphs $\mathscr{C}$, then the 3-colorability problem, or the hamiltonicity problem are polynomial-time solvable on $\mathscr{C}$, since both those problems can be expressed using an MSO_2 sentence, whose quantifiers that range over *sets* of vertices and edges of the graph.

There has been a long line of work on fixed-parameter tractability of model-checking FO, as it is arguably the most fundamental logic. This culminated in the work of Grohe, Kreutzer and Siebertz [24], who proved that this problem is fixed-parameter tractable on every class that is *nowhere dense*. Those classes include for example the class of planar graphs, or every class with bounded maximum degree, or of bounded genus. Without going into details, nowhere denseness is a general notion of uniform sparseness in graphs, and it is broader than most well-studied concepts of sparseness considered in structural graph theory, such as excluding a fixed (topological) minor. As observed by Dvořák et al. [19], the result of Grohe et al. is tight in the following sense: whenever $\mathscr{C}$ is not nowhere dense and is closed under taking subgraphs, then FO model-checking on $\mathscr{C}$ is as hard as on general graphs, that

is, AW[⋆]-hard. This means that as far as subgraph-closed classes are concerned, sparsity – described formally through the notion of nowhere denseness – exactly delimits the area of tractability of FO and the limits of the locality method for FO model-checking.

The aforementioned classic theorem of Courcelle [13] states that the model-checking problem for MSO_2 is fixed-parameter tractable on a class $\mathscr{C}$ if $\mathscr{C}$ has bounded treewidth. The proof translates the given sentence into a suitable tree automaton working on a tree decomposition of the input graph; this approach brings a wide range of automata-based tools to the study of MSO_2 on graphs. Again, as shown by Kreutzer and Tazari [31] and Ganian et al. [23], under certain complexity assumptions this is (almost) the most one could hope for: on subgraph-closed classes whose treewidth is poly-logarithmically unbounded, model-checking of MSO_2 becomes intractable from the parameterized perspective. So for MSO_2, bounded treewidth (almost) delimits the frontier of tractability, at least for subgraph-closed classes, and automata-based techniques exactly explain what can be done algorithmically.

With the aim to provide a meta theorem that captures the essence of the techniques developed to answer connectivity under vertex failures, we search for a logic that can express these queries. It is easy to see that MSO_2 is strong enough, but in fact, MSO_2 is much stronger and as explained above, it is intractable beyond graphs of bounded treewidth. On the other hand, FO can only express local properties and in particular, it cannot even express the very simple query of whether two vertices are in the same connected component. This naturally leads to an extension of FO that may use connectivity under vertex failures as atomic formulas. Such a logic was very recently introduced independently by Bojańczyk [6] and by Schirrmacher et al. [39] under the name *separator logic*, and denoted $\mathsf{FO}+\mathsf{conn}$. This logic extends FO by predicates $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ (one for each $k \geqslant 0$) with the following semantics: if $s,t,u_1,\ldots,u_k$ are vertices of a graph G, then $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ holds if and only if there is a path that connects s with t and does not pass through any of the vertices $u_1,\ldots,u_k$. Thus, separator logic involves very basic connectivity queries that have a global character, unlike plain FO, which is local. As a consequence, separator logic can express many interesting algorithmic properties, such as acyclicity, k-connectivity, the feedback vertex set problem parameterized by solution size and many recently studied elimination distance problems [9]. On the other hand, while $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ predicates are expressible in MSO_2, it is no surprise that the expressive power of $\mathsf{FO}+\mathsf{conn}$ is strictly weaker than that of MSO_2. For example, MSO_2 can express bipartiteness, while $\mathsf{FO}+\mathsf{conn}$ cannot [39, Theorem 3.11]. Separator logic is also incomparable with the recently introduced compound logic [20]. While compound logic can express the atomic connectivity predicates under vertex failures, it is not closed under negation. On the other hand, compound logic can express the exclusion of minors, a property not expressible in separator logic.

As the second main result of the paper, we prove an algorithmic meta theorem for separator logic. We show that the frontier of tractability of $\mathsf{FO}+\mathsf{conn}$ is almost exactly delimited by classes of graphs that exclude a fixed topological minor. More precisely, we prove the following complementary results.

▶ **Theorem 1.2.** *Let $\mathscr{C}$ be a class of graphs that exclude a fixed graph as a topological minor. Then given $G \in \mathscr{C}$ and an $\mathsf{FO}+\mathsf{conn}$ sentence φ, one can decide whether φ holds in G in time $f(\varphi)\cdot\|G\|^3$, where f is a computable function depending on $\mathscr{C}$.*

Our result implies e.g. the recent result about elimination distance to bounded degree [3], but it is much more general as it unifies in particular all elimination distance problems to classes definable in separator logic on classes that exclude a topological minor. Note that it does not subsume the result of [27] as e.g. the elimination distance to bipartite graphs and vertex planarization cannot be expressed by separator logic.

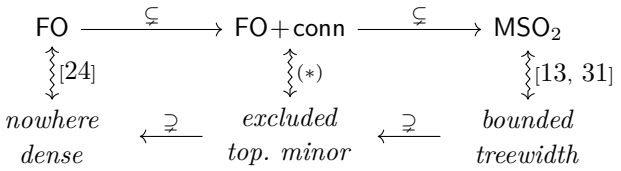

Figure 1 Correspondences between logics and properties of graph classes. An arrow $\mathcal{L} \rightsquigarrow \mathcal{P}$ between a logic $\mathcal{L}$ and a property $\mathcal{P}$ of graph classes denotes a correspondence: the logic $\mathcal{L}$ is tractable on a graph class $\mathscr{C}$ if and, under additional assumptions, only if, the class $\mathscr{C}$ has the property $\mathcal{P}$. Our result is the central correspondence $(*)$.

For the lower bound, we will need a technical assumption. We say that a class $\mathscr{C}$ admits *efficient encoding of topological minors* if for every graph H there exists $G \in \mathscr{C}$ such that H is a topological minor of G, and, given H, such G together with a suitable topological minor model can be computed in time polynomial in $|H|$. Note that in particular such G can be only polynomially larger than H.

▶ **Theorem 1.3.** *Let $\mathscr{C}$ be a subgraph-closed class of graphs that admits efficient encoding of topological minors. Then the model-checking of* $\mathsf{FO}{+}\mathsf{conn}$ *on $\mathscr{C}$ is* $\mathsf{AW}[\star]$*-hard, even for the fragment that uses only* conn_k *predicates with $k \leqslant 2$.*

On the one hand, these results show that separator logic is a natural logic whose expressive power lies strictly between FO and MSO_2, and which corresponds, in the sense described above, to a natural property of classes of graphs that lies between bounded treewidth and nowhere denseness, see Figure 1. On the other hand, the proof of Theorem 1.2 provides a general meta-explanation of the technique of dynamic programming over tree decompositions with unbreakable parts. In particular, the tractability result provided by Theorem 1.2 generalizes several recent algorithmic results for concrete problems expressible in separator logic [3, 10, 32]. However, there are some concrete problems where the general methodology non-trivially applies and which seem not to be captured by our meta-theorem [1, 27].

The proof of Theorem 1.3 is easy (see Section 3), so we focus on sketching the proof of Theorem 1.2, which is the main technical contribution of the second part of the paper. Let us fix an $\mathsf{FO}{+}\mathsf{conn}$ sentence φ, and let $k+2$ be the maximum arity of connectivity predicates appearing in φ. Again, the key observation is that conn_k predicates can be rewritten to plain FO provided the graph is well-connected, and the reason for this is the same combinatorial observations that underlies the proof of Theorem 1.1. Coming back to our dynamic programming on the tree decomposition, we choose to present it using a new framework based on automata, as this brings us closer to the classic understanding of logic on tree-decomposable graphs. Intuitively, the automaton processes a given tree in a bottom-up manner, but we assume that on the children of every node there is an additional structure of a graph. We say that such trees are *augmented* with graphs. When the automaton processes a node x, it chooses a transition based on an FO query executed on the graph on the children of x, where each child is labeled with the state computed for it before in the run. Then testing whether a formula $\varphi \in \mathsf{FO}{+}\mathsf{conn}$ holds on a graph G is reduced to deciding whether an automaton constructed from φ accepts such an *augmented tree*, constructed from the tree decomposition provided by the theorem of Cygan et al. [15]. The run of the automaton can be computed efficiently when the graphs augmenting the tree admit efficient FO model-checking, which is the case when the input graph excludes a fixed topological minor.

The benefit of this approach is that other questions related to the logic FO+conn can be reduced in the same way to questions about tree automata over augmented trees. We showcase this by proving that given an FO+conn query $\varphi(\bar{x})$ with free variables and a graph G from a fixed topological-minor-free class $\mathscr{C}$, the answers to $\varphi(\bar{x})$ on G can be enumerated with constant delay after fpt preprocessing (see Theorem 8.2 of the full version for a precise statement). Similarly, the formula $\varphi(\bar{x})$ can be queried in constant time after fpt preprocessing (see Theorem 8.1 of the full version). In general, we believe that the framework of automata over augmented trees may be of independent interest, as it seems to be a convenient model for understanding dynamic programming procedures over tree-decomposable structures.

Similar fpt query-answering and enumeration algorithms for plain FO have been obtained for classes with bounded expansion [30, 41], low degree [18], and nowhere dense classes [40] and for MSO_2 on trees and graphs of bounded treewidth [5, 29, 4]. Our general approach based on augmented trees allows us to generalize those results to trees that are augmented with graphs coming from a nowhere dense class, and to a certain logic combining the power of MSO on trees and FO on graphs.

Let us comment on the novelty of using the decomposition theorem of Cygan et al. [15]. The result of Cygan et al. [15] has been used several times for various graph problems [14, 15, 35, 33, 38]. Our application is the most similar to (and in fact, draws inspiration from) the work of Lokshtanov et al. [34], who proved the following statement: for every CMSO_2 sentence φ and $k \in \mathbb{N}$, model-checking φ on general graphs can be reduced to model-checking φ on (q,k)-unbreakable graphs, where q is a constant depending on φ and k. As FO+conn is subsumed by CMSO_2, this result can be almost applied to establish Theorem 1.2. The caveat is that the unbreakable graph output by the reduction needs to admit efficient FO model-checking so that the considered FO+conn sentence can be decided in fpt time after rewriting it to plain FO. In essence, we show that it is possible to guarantee this provided the input graph excludes a fixed topological minor. We remark that the work of Lokshtanov et al. [34] does not use the decomposition theorem of Cygan et al. [15] directly, but relies on its conceptual predecessor, the *recursive understanding* technique [11, 28].

Organization. In the extended abstract we give a detailed overview over the proofs of our main results: Theorem 1.1, Theorem 1.2, and Theorem 1.3. We provide all formal definitions and proofs in the full version [36]. The theorem numbers for references in the full version are provided in parenthesis.

2 Connectivity under vertex failures: overview of Theorem 1.1

In this section we provide an overview of the proofs of Theorem 1.1. This proof exposes all main ideas in a purely algorithmic setting. Then we discuss how the same methodology can be used for Theorem 1.2. In this introduction we assume familiarity with basic terminology of tree decompositions.

Unbreakability. As mentioned in Section 1, the central idea of this work is to tackle problems involving connectivity predicates using a decomposition into well-connected – or, more formally, *unbreakable* – parts. We first need to recall a few definitions.

A *separation* in a graph G is a pair (A,B) of vertex subsets such that $A \cup B = V(G)$ and there are no edges in G between $A - B$ and $B - A$. The *order* of the separation is the cardinality of the *separator* $A \cap B$. A vertex subset X is (q,k)-*unbreakable* in a graph G if for every separation (A,B) in G of order at most k in G, either $|A \cap X| \leqslant q$ or $|B \cap X| \leqslant q$.

So intuitively speaking, a separation of order k cannot break X in a balanced way: one of the sides must contain at most q vertices of X. For example, cliques and $k+1$-connected graphs are (k,k)-unbreakable, and square grids are $(\mathcal{O}(k^2),k)$-unbreakable.

The notion of unbreakability has been implicitly introduced in the context of parameterized algorithms by Kawarabayashi and Thorup in their work on the k-Way Cut problem [28]. This work has brought about the method of *recursive understanding*, which was then made explicit by Chitnis, Cygan, Hajiaghayi, Pilipczuk, and Pilipczuk in the technique of *randomized contractions* [11]. Intuitively, recursive understanding is a Divide&Conquer scheme using which one can reduce problems on general graphs to problems on suitably unbreakable graphs, provided certain technical conditions are satisfied. The technique was also applied in the context of model-checking: Lokshtanov, Ramanujan, Saurabh, and Zehavi [34] proved that the problem of deciding a property expressed in CMSO_2 in fpt time on general graphs can be reduced to the same problem on suitably unbreakable graphs. This result was used to provide algorithms for computing elimination distance to certain graph classes [2, 21, 26], which is very much related to separator logic (see [39, Example 3.5]).

In this work, we will not use recursive understanding or randomized contractions *per se*, but their conceptual successor: the decomposition into unbreakable parts. Precisely, the following theorem was proved by Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [15].

▶ **Theorem 2.1** ([15]). *There is a function $q(k) \in 2^{\mathcal{O}(k)}$ such that given a graph G and $k \in \mathbb{N}$, one can in time $2^{\mathcal{O}(k^2)} \cdot |G|^2\|G\|$ construct a (rooted) tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ of G satisfying the following.*

- *All adhesions in $\mathcal{T}$ are of size at most $q(k)$; and*
- *every bag of $\mathcal{T}$, say at node x, is $(q(k),k)$-unbreakable in the subgraph of G induced by the union of bags at all descendants[2] of x.*

We remark that a different variant of Theorem 2.1 was later proved by Cygan, Komosa, Lokshtanov, Pilipczuk, Pilipczuk, and Wahlström [14]. This variant provides much stronger unbreakability guarantees – $q(k) = k$ – at the cost of relaxing the unbreakability to hold only in the whole graph G. See Section 3 of the full version for a discussion. The variant of [14] can be also used in our context and this leads to improving some quantitative bounds, however for simplicity, we focus on Theorem 2.1 in this overview.

A typical usage of Theorem 2.1 is to apply bottom-up dynamic programming on the obtained tree decomposition $\mathcal{T} = (T, \mathsf{bag})$. The subproblem for a node x of T corresponds to finding partial solutions in the subgraph of G induced by the union of bags at descendants of x, for every possible behavior of such a partial solution on the adhesion connecting x with its parent. That this adhesion has size at most $q(k)$ gives an upper bound on the number of different behaviors. To solve such a subproblem one needs to aggregate the solutions computed for the children of x by solving an auxiliary problem on the subgraph induced by the bag of x. In various problems of interest, the unbreakability of the bag becomes helpful in solving the auxiliary problem. This general methodology has been successfully used to give multiple fpt algorithms for cut problems [15, 14, 38], most prominently for Minimum Bisection [15]. More recent uses include a parameterized approximation scheme for the Min k-Cut problem [35] and a fixed-parameter algorithm for Graph Isomorphism parameterized by the size of the excluded minor [33]. On a high level, we apply the same methodology here.

[2] We follow the convention that every node is its own ancestor and descendant.

The case of totally unbreakable graphs. Let us come back to the problem of evaluating connectivity queries: we are given a graph G and after some preprocessing of G, we would like to be able to quickly evaluate for given vertices $s, t, u_1, \ldots, u_k$ whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. Consider first the following special case: the whole G is (q, k)-unbreakable for some parameter q (or more formally, $V(G)$ is (q, k)-unbreakable in G). The key observation is that then, whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds can be easily decided in time polynomial in q and k as follows.

Run a breadth-first search from s in $G - \{u_1, \ldots, u_k\}$, but terminate the search once $q+1$ different vertices have been reached. If the search reached t, then for sure $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. If the search finished before reaching $q+1$ different vertices and it did not reach t, then for sure $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ does not hold. In the remaining case, perform a symmetric procedure starting from t. The key observation is that if both breadth-first searches – from s and from t – got terminated after reaching $q+1$ different vertices, then the (q, k)-unbreakability of G implies that $\{u_1, \ldots, u_k\}$ do not separate s from t. So then $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. It is straightforward to implement this procedure in time polynomial in q and k given the standard encoding of G through adjacency lists.

In the general case, we cannot expect the given graph G to be (q, k)-unbreakable for any parameter q bounded in terms of k. However, we can use Theorem 2.1 to compute a tree decomposition $\mathcal{T}$ of G into parts that are (q, k)-unbreakable for $q \in 2^{\mathcal{O}(k)}$. The idea is that then, a query $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ can be evaluated by bottom-up dynamic programming on $\mathcal{T}$. By suitably precomputing enough auxiliary information about $\mathcal{T}$, this evaluation can be performed in time depending only on q and k.

Evaluating a query on the decomposition. Consider then the following setting: we are given the decomposition $\mathcal{T} = (T, \mathsf{bag})$ provided by Theorem 2.1, and for given $s, t, u_1, \ldots, u_k$ we would like to decide whether $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ holds. So far let us not optimize the complexity: the goal is only to design a general algorithmic mechanism which will be later implemented efficiently using appropriate data structures.

For every node x of T, let $\mathsf{adh}(x)$ be the adhesion between x and its parent (or $\emptyset$ if x is the root), let $\mathsf{cone}(x)$ be the union of the bags at the descendants of x, and let $\mathsf{comp}(x) = \mathsf{cone}(x) - \mathsf{adh}(x)$. Call a node x *affected* if $\mathsf{comp}(x)$ contains a vertex of $\{s, t, u_1, \ldots, u_k\}$, and *unaffected* otherwise. Further, let $D(x) = \mathsf{adh}(x) \cup (\mathsf{comp}(x) \cap \{s, t\})$; note that $|D(x)| \leqslant q + 2$. Our goal is to compute the following *(connectivity) profile* for every node x of T:

$$\mathsf{profile}(x) = \left\{ \{a, b\} \in \binom{D(x)}{2} \;\middle|\; a \text{ and } b \text{ are connected in } G[\mathsf{cone}(x) - \{u_1, \ldots, u_k\}] \right\}.$$

We remark that the tree decomposition $\mathcal{T}$ can be chosen so that it satisfies the following basic connectivity property: for every node x and vertices $a, b \in \mathsf{adh}(x)$, there is a path connecting a and b whose all internal vertices belong to $\mathsf{comp}(x)$. Thus, for every unaffected node x, we have that $\mathsf{profile}(x) = \binom{D(x) - \{u_1, \ldots, u_k\}}{2}$.

It is easy to see that the profile of x is uniquely determined by the profiles of the children of x and the subgraph induced by the bag of x. The following lemma shows that this can be done efficiently.

▶ **Lemma 2.2** (informal statement, Lemma 4.4 in the full version). *Let x be a node of T. Suppose that for each affected child z of x we are given* $\mathsf{profile}(z)$*. Then, subject to suitable preprocessing of G, one can compute* $\mathsf{profile}(x)$ *in time polynomial in q.*

Note that every node has at most $k+2$ affected children, hence the input to the algorithm of Lemma 2.2 is of size polynomial in q.

Let us sketch the proof of Lemma 2.2. In essence, we apply the same strategy as the one from the case when the whole graph G is unbreakable. We define the *bag graph of* x, denoted $\mathsf{bgraph}(x)$, as the graph[3] obtained from $G[\mathsf{bag}(x)]$ by turning the adhesion $\mathsf{adh}(z)$ into a clique, for every child z of x. Note that $\mathsf{bgraph}(x)$ does not depend on the query, and thus can be precomputed upon initialization. As $\mathsf{bag}(x)$ is (q,k)-unbreakable in $G[\mathsf{cone}(x)]$, to test whether two vertices $a, b \in \mathsf{bag}(x)$ are connected in $G[\mathsf{cone}(x) - \{u_1, \ldots, u_k\}]$ it suffices to apply breadth-first searches from a and b in $\mathsf{bgraph}(x)$ that are terminated after reaching $q+1$ different vertices, except that these searches are forbidden to use vertices from $\{u_1, \ldots, u_k\} \cap \mathsf{bag}(x)$ as well as those edges originating from contracting the adhesion of affected children into cliques that are not in respective profiles. Provided $\mathsf{bgraph}(x)$ is precomputed and a list of affected children together with their profiles is given on input, it is easy to implement this algorithm so that it runs in time polynomial in q. Thus we can decide for every pair $\{a,b\} \in \binom{\mathsf{adh}(x)}{2}$ whether it should be included in $\mathsf{profile}(x)$. A similar reasoning can be applied to pairs in $\binom{D(x)}{2}$ that include s or t.

By applying Lemma 2.2 bottom-up, we can compute all the profiles in time $q^{\mathcal{O}(1)} \cdot |T| \leqslant q^{\mathcal{O}(1)} \cdot |G|$. Then we can read whether $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ holds by checking whether $\{s,t\}$ belongs to the profile of the root of T. Hence, the query $\mathsf{conn}_k(s,t,u_1,\ldots,u_k)$ can be evaluated in time $q^{\mathcal{O}(1)} \cdot |G|$.

Data structure. Our final goal is to enrich the decomposition $\mathcal{T}$ with some additional information so that the mechanism presented above can be executed in time depending only on q.

For a vertex v of G, let $\mathsf{top}(v)$ be the unique top-most node of T whose bag contains v. Let $\widehat{S} := \{s,t,u_1,\ldots,u_k\}$ and let $X = \{\mathsf{top}(v) \colon v \in \widehat{S}\}$.

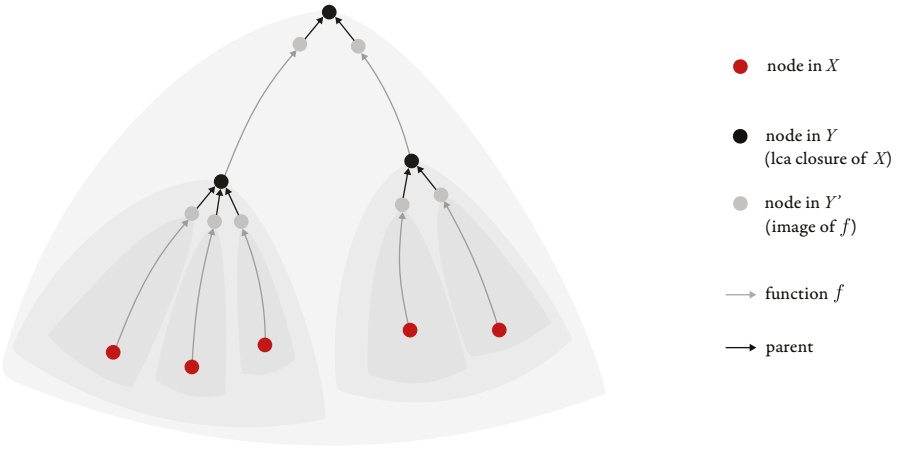

Figure 2 The sets X, Y and Y'. Only the nodes from $Y \cup Y'$ are marked in the figure, wheras the shaded areas indicate subtrees consisting of nodes which are not marked in the figure. Every affected node lies on a path from some $y \in Y$ to $f(y) \in Y'$.

[3] The bag graph $\mathsf{bgraph}(x)$ is actually defined slightly differently for technical reasons, as the graph obtained from $G[\mathsf{cone}(x)]$ by contracting, for each $y \in \mathsf{children}(x)$, the set $\mathsf{comp}(y)$ into a single vertex, identified with y. See the full version for the details.

Note that $|X| \leqslant |\widehat{S}| \leqslant k+2$ and that a node of T is affected if and only if it is an ancestor of a node from X. Let now Y be the *lowest common ancestor closure* of X: the set comprising X and all lowest common ancestors of pairs of vertices of X. It is well-known that then $|Y| \leqslant 2|X| - 1 \leqslant 2k+3$. We also add the root of T to Y in case it is not already present; thus $|Y| \leqslant 2k+4$. Observe that if for every vertex v we store a pointer to $\mathsf{top}(v)$, and we set up the data structure for lowest common ancestor queries of Harel and Tarjan [25] on T, then for given $s, t, u_1, \ldots, u_k$ the set Y can be computed in time polynomial in k. For every node y in Y let $f(y)$ denote the closest ancestor of y whose parent belongs to Y, or y if no such ancestor exists (see Figure 2). Note that every affected node lies on a path joining y with $f(y)$, for some $y \in Y$. Let $Y' = \{f(y) \mid y \in Y\}$. Then Y' is the set of all nodes that are affected children of a node in Y. Note that every element in Y' has exactly one preimage under f, that is, there is an function $f^{-1}\colon Y' \to Y$ such that $f(f^{-1}(y')) = y'$ for all $y' \in Y'$. In particular, $|Y'| \leqslant |Y|$. Using a slight modification of the data structure of Harel and Tarjan [25], we can compute Y' as well as the function $f^{-1}\colon Y' \to Y$ in time polynomial in k.

The idea is that when evaluating query $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$, we can compute the profiles only for the nodes of $Y \cup Y'$, instead of all affected nodes of T. Note that the root of T was explicitly added to Y, so we will still be able to read the answer to $\mathsf{conn}_k(s, t, u_1, \ldots, u_k)$ from the profile of the root.

We process the nodes of $Y \cup Y'$ in a bottom-up manner. Consider then any element $x \in Y \cup Y'$; our task is to compute its profile based on the profiles of its strict descendants belonging to $Y \cup Y'$. We distinguish two cases. In the first case, $x \in Y$, so we may compute $\mathsf{profile}(x)$ using the algorithm of Lemma 2.2. The second case is of an element $x \in Y'$; then $x = f(y)$ where $y = f^{-1}(x)$. Observe that the path from y to x consists of affected nodes, whose siblings are unaffected nodes. We now show how $\mathsf{profile}(x)$ can be computed from $\mathsf{profile}(y)$.

For two nodes c, d of T, where c is an ancestor of d, let $\mathsf{torso}(c, d)$ be the set of all pairs $\{a, b\} \in \binom{\mathsf{adh}(c) \cup \mathsf{adh}(d)}{2}$ such that in G there is a path from a to b whose all internal vertices belong to $\mathsf{comp}(c) - \mathsf{cone}(d)$. Then to compute $\mathsf{profile}(x)$ from $\mathsf{profile}(y)$, construct an auxiliary graph with vertices $(\mathsf{adh}(x) \cup \mathsf{adh}(y)$ and edges $\mathsf{profile}(y) \cup \mathsf{torso}(x, y)$. Now observe that $\mathsf{profile}(x)$ is the reachability relation in this graph restricted to $\mathsf{adh}(x) - \{u_1, \ldots, u_k\}$. The key point implying the correctness of this observation is that there are no vertices from X in $\mathsf{comp}(x) - \mathsf{comp}(y)$.

Observe now that the values of $\mathsf{torso}(c, d)$ for (c, d) ranging over ancestor/descendants pairs in T are independent of the query. Therefore, upon initialization we can compute a data structure that can be queried for those valued efficiently. The statement below describes two possible implementations.

▶ **Lemma 2.3** (Lemmas 4.2 and 6.2 of the full version, combined and improved). *Given $\mathcal{T}$, one can set up data structures that can answer $\mathsf{torso}(c, d)$ queries with the following specifications:*

1. *initialization time $q^{\mathcal{O}(1)} \cdot |G|^2\|G\|$, memory usage $q^{\mathcal{O}(1)} \cdot |G|^2$, query time $q^{\mathcal{O}(1)}$; or*
2. *initialization time $2^{\mathcal{O}(q^2)} \cdot |G|$, memory usage $2^{\mathcal{O}(q^2)} \cdot |G|$, query time $q^{\mathcal{O}(1)}$.*

The first point of Lemma 2.3 is actually straightforward: just compute and store all the $\mathcal{O}(|T|^2) = \mathcal{O}(|G|^2)$ answers to the torso queries, each in time $q^{\mathcal{O}(1)} \cdot \|G\|$ using $q^{\mathcal{O}(1)}$ applications of breadth-first search. The second point, which trades exponential dependency on q for obtaining linear memory usage and initialization time, is more interesting. Before we discuss it, let us observe that we may now complete the proof of Theorem 1.1. Indeed, using the data structure from the second point of Lemma 2.3, we can compute $\mathsf{profile}(x)$

from $\mathsf{profile}(y)$ for each $x \in Y'$ and $y = f^{-1}(x)$. By performing this procedure for all the at most $2(2k+4)$ nodes of $Y \cup Y'$ in a bottom-up manner, we eventually compute the profile of the root of T, from which the answer to the query can be read. The running time is dominated by $q^{\mathcal{O}(1)}$ calls to the data structure of the second point of Lemma 2.2, each taking $2^{\mathcal{O}(q^2)} = 2^{2^{\mathcal{O}(k)}}$ time.

Finally, let us discuss the proof of the second point of Lemma 2.3. We reduce this problem to the problem of evaluating *product queries* in a semigroup-labeled tree, defined as follows. Suppose T is a rooted tree whose edges are labeled with elements of a finite semigroup S. The task is to set up a data structure that for given nodes x, y, where x is an ancestor of y, outputs the product of the elements of S on the path in T from x to y. The problem of evaluating torso queries can be reduced to this abstract setting by considering a suitable semigroup of *bi-interface graphs*, which represent connectivity between corresponding adhesions; see [7, 8] for the origins of this concept. This semigroup has size $2^{\mathcal{O}(q^2)}$. So it then suffices to use the following result.

▶ **Theorem 2.4** (Przybyszewski). *There is a data structure for the problem of evaluating product queries in a tree T labeled with elements of a finite semigroup S that achieves query time $\log(|S|)^{\mathcal{O}(1)}$, memory usage $|S|^{\mathcal{O}(1)} \cdot |T|$, and initialization time $|S|^{\mathcal{O}(1)} \cdot |T|$.*

Note that this formulation is an improvement over Theorem 5.1 in the first version of our paper [36], as the query time $|S|^{\mathcal{O}(1)}$ is replaced by $\log(|S|)^{\mathcal{O}(1)}$. This improvement is due to Wojciech Przybyszewski.

This finishes the proof of Theorem 1.1. Note that in the argumentation there are two modules that can be replaced with other solutions:

- The decomposition of Theorem 2.1 can be replaced with the more recent variant from [14]. This allows us to have $q = k$, implying that all the $2^{2^{\mathcal{O}(k)}}$ factors are replaced with $2^{\mathcal{O}(k^2)}$. The cost is increasing the polynomial factor of the initialization time to an unspecified term $|G|^{\mathcal{O}(1)}$ and several technical complications in the analysis, which nevertheless can be overcome.
- Instead of using the data structure of the second point of Lemma 2.3, we can use the first point. This allows us to have polynomial dependency on q in the query time and space usage at the cost of quadratic dependence on $|G|$ in the memory usage.

In particular, if only the first replacement is performed, then we obtain a data structure with initialization time $2^{\mathcal{O}(k^2)} \cdot |G|^{\mathcal{O}(1)}$, memory usage $2^{\mathcal{O}(k^2)} \cdot \|G\|$, and query time $k^{\mathcal{O}(1)}$, and if both are performed, then we obtain a data structure with initialization time $2^{\mathcal{O}(k \log k)} \cdot |G|^{\mathcal{O}(1)}$, memory usage $k^{\mathcal{O}(1)} \cdot |G|^2$, and query time $k^{\mathcal{O}(1)}$.

3 Model-checking FO+conn: Theorems 1.2 and 1.3

We now turn to the proofs of Theorem 1.2 and Theorem 1.3. In those theorems, we fix a class $\mathscr{C}$ and consider the model-checking problem for the logic $\mathsf{FO}{+}\mathsf{conn}$, that is, the problem of deciding whether a given $\mathsf{FO}{+}\mathsf{conn}$ sentence holds in a given graph $G \in \mathscr{C}$.

Lower bound. First let us explain why we require $\mathscr{C}$ to exclude a fixed topological minor in Theorem 1.2. Clearly, the model-checking problem for $\mathsf{FO}{+}\mathsf{conn}$ generalizes the model-checking problem for FO, so $\mathscr{C}$ should be in particular a class for which it is known that FO model-checking is FPT. This is the case not only for classes that exclude a topological minor but also for all nowhere dense classes [24]. So let us see why it is not enough to merely assume that $\mathscr{C}$ is nowhere dense.

For a graph G and $k \in \mathbb{N}$, the *k-subdivision* $G^{(k)}$ of G is obtained from G by replacing each edge by a path of length $k+1$. It follows from the definition of nowhere denseness that the class $\mathscr{C}$ of all graphs of the form $G^{(n)}$, where G is an arbitrary graph on n vertices, is nowhere dense. However, a formula of $\mathsf{FO}+\mathsf{conn}$ can easily recover G from $G^{(n)}$, at least when G has no vertices of degree 2. Indeed, the original vertices of G are precisely the vertices of $G^{(n)}$ that have degree at least 3, which can be expressed by an FO formula $\varphi_V(x)$. Furthermore, two such vertices u, v are adjacent in G if and only if there is some vertex z of degree 2 in $G^{(n)}$ such that for every vertex w of $G^{(n)}$, $\mathsf{conn}_2(z, w, u, v)$ holds only if w has degree 2 in $G^{(n)}$. This condition can be written using an $\mathsf{FO}+\mathsf{conn}$ formula $\varphi_E(u, v)$. Using this observation, any FO sentence φ can be replaced by an $\mathsf{FO}+\mathsf{conn}$ sentence φ', so that for every graph G as above, φ holds in G if and only if φ' holds in $G^{(n)}$. Essentially, φ' is obtained from φ by replacing every atomic formula $E(x, y)$ (denoting adjacency) by $\varphi_E(x, y)$, and guarding all quantifiers by $\varphi_V(x)$. Therefore, if we could efficiently model-check φ' on the nowhere dense class $\mathscr{C}$, we could as well model-check φ on the class of all graphs (of size at least 3 and minimum degree at least 3), as follows: given an arbitrary graph G, first construct the graph $G^{(n)} \in \mathscr{C}$ (in time polynomial in $n = |V(G)|$) and then test φ' on $G^{(n)}$. Hence, if $\mathsf{FO}+\mathsf{conn}$ model-checking was fpt on $\mathscr{C}$, then FO model-checking would be fpt on the class of all graphs, implying $\mathsf{FPT} = \mathsf{AW}[\star]$. So nowhere dense graph classes are too general for the statement of Theorem 1.2 to hold. Intuitively, the notion of nowhere denseness is not preserved under contracting long paths, which can be simulated in the logic $\mathsf{FO}+\mathsf{conn}$.

The same simple argument proves Theorem 1.3. Indeed, the argument works not only for the graph $G^{(n)}$ obtained as the n-subdivision of G but also for any subdivision G' (obtained by replacing each edge independently by any number of vertices) of G, as long as G' can be computed from G in polynomial time. The condition that $\mathscr{C}$ is subgraph-closed and admits efficient encoding of topological minors guarantees precisely that for any given graph G we can construct, in polynomial time, a graph G' which is a subdivision of G and belongs to $\mathscr{C}$. This yields Theorem 1.3 (see Section 9 of the full version for details).

Upper bound. We now give some details concerning the proof of Theorem 1.2, which is the second main contribution of the paper.

The starting observation is that for a (q, k)-unbreakable graph G, the query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ can be expressed in plain FO. Indeed, this query fails if and only if there is a set A of at most q vertices which contains exactly one of the vertices u and v, and such that all neighbors of vertices of A outside of A are contained in $\{x_1, \ldots, x_k\}$. The existence of such a set of q vertices can be expressed using q existential quantifiers followed by a universal quantifier. So $\mathsf{FO}+\mathsf{conn}$ with connectivity queries of arity at most $k+2$ is equally expressive as plain FO on (q, k)-unbreakable graphs, as long as q is a constant.

Now, if the graph G is not (q, k)-unbreakable, we work with the tree decomposition into (q, k)-unbreakable parts given by Theorem 2.1, where $q = q(k)$ is a constant depending on k. The next observation is that the constant-time querying algorithm used in the proof of Theorem 1.1 and explained above, can be seen as a formula on a suitably defined logical structure. Indeed, if we look into the data structure answering the queries $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$, we notice that each such query triggers a constant number of operations asking about the least common ancestor of two nodes in the tree decomposition, or asking about the membership of a vertex to a bag. Additionally, the algorithm may ask, for two vertices u, v belonging to a bag x, whether u and v are adjacent in the original graph, or whether they are adjacent in the bag graph $\mathsf{bgraph}(x)$. Finally, the algorithm also performs torso queries. We may therefore construct a logical structure, corresponding to the data structure, such that this

basic functionality is expressible by a first-order formula in this structure. Then the query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ becomes expressible using a plain FO formula in the logical structure. It will then remain to show that we can model check FO formulas in fpt on the resulting logical structure.

It is convenient to abstract away the details of our logical structure, and represent it in the following form: It is a tree T (possibly vertex-labeled), together with additional edges (possibly labeled) between some nodes which are siblings in the tree. We call such a structure an *augmented tree*. In such a tree, the set of children of any node v carries a structure of a graph. In the particular case of our construction, those graphs will precisely correspond to bag graphs. Recall that a bag graph is obtained from the subgraph induced by a bag by turning all adhesions towards children into cliques. It is not difficult to show (see Lemma 3.4 of the full version) that if the original graph G is H-topological-minor free, then the resulting bag graphs are H'-topological-minor-free for some H' depending on H and on k.[4] Hence, we obtain a tree augmented with H'-topological-minor-free graphs. And the original query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ can be now represented as an FO formula in the augmented tree, where the FO formula may use the ancestor relation $\leqslant$ on the tree nodes, as well as the adjacency relation between the siblings. Showing this basically amounts to revisiting the proof of Lemma 2.2 and arguing that the atomic algorithmic operations can now be simulated by FO formulas in the augmented tree.

Let us note here that the remainder of the argument would go through even if we assumed that the resulting tree is augmented with graphs from a fixed nowhere dense class $\mathscr{C}'$ which is not necessarily topological-minor-free. However, if the original graph G merely belongs to a nowhere dense class $\mathscr{C}$, then the bag graphs, and hence the graphs in the augmented tree, no longer need to belong to any fixed nowhere dense class. In fact, they may contain arbitrarily large cliques.

Continuing with the proof, now it remains to show that we can solve FO model-checking on trees augmented with H'-topological-minor-free graphs in fixed-parameter time, where the FO formulas may use the ancestor relation in the tree as well as the edge relation among the siblings. For this, we lift the usual, automata-based techniques for model-checking on trees, to the case of augmented trees. We define automata that process augmented trees in a bottom-up fashion, labeling the nodes of the tree with states from a finite set of states, starting from the leaves and proceeding towards the root. At any node x, to determine the state of the automaton at x, we consider the graph induced on the children of x in the augmented tree, vertex-labeled by the states computed earlier. Then the state at x is determined by evaluating a fixed collection of first-order sentences on this labeled graph. In this way, all nodes of the tree are labeled by states, and the automaton accepts the augmented tree depending on the state at the root.

By construction, it can be decided in fpt time whether or not a given such automaton accepts a given tree that is augmented with graphs that come from, say, a nowhere dense class. Additionally, using standard techniques, we show that for every first-order sentence φ on augmented trees there is such an automaton that determines whether φ holds in a given augmented tree. Therefore, FO model-checking is fpt on trees augmented with graphs from a nowhere dense class; this in particular applies to classes with excluded topological minors.

[4] Here we refer to the bgraph as defined in Footnote 3.

To summarize, to model-check a sentence φ of $\mathsf{FO}+\mathsf{conn}$ on a H-topological-minor-free graph G, we perform the following reasoning:

1. Using Theorem 2.1, compute a tree decomposition $\mathcal{T}$ of G with (q,k)-unbreakable bags, where $k+2$ is the maximal arity of the connectivity predicates in φ, and $q = q(k)$.
2. Turn $\mathcal{T}$ into a tree $\mathcal{T}'$ augmented with H'-topological-minor free graphs, where H' depends on H and k.
3. In the augmented tree $\mathcal{T}'$, the connectivity predicate $\mathsf{conn}_k(u,v,x_1,\ldots,x_k)$ can be expressed using a plain FO formula. This follows by analyzing how connectivity predicates are evaluated in constant time, and observing that the atomic operations can be performed using FO formulas in the augmented trees.
4. Therefore, also φ can be expressed as a plain FO formula φ', which is evaluated in $\mathcal{T}'$.
5. Convert φ' into an automaton $\mathcal{A}$ processing augmented trees, using standard automata-based techniques.
6. Run $\mathcal{A}$ on $\mathcal{T}'$ in a bottom-up fashion. This involves performing FO queries on the graphs induced on the children of any given node, and uses one of the known algorithms for FO model-checking on H'-topological-minor-free graphs.

In Section 7 of the full version we describe the basic toolbox of automata on augmented trees, explaining in detail the last two steps above, abstracting away from the specific application, and providing further results on augmented trees, e.g. concerning query enumeration. In fact, with the same methods we can show that not only FO model-checking is fpt on trees augmented with graphs from a nowhere dense class, but also a more general logic $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A) \cup \Sigma)$ combining the power of FO on the augmenting graphs with MSO on trees can be solved in fpt on augmented trees. We then show in Section 8 (of the full version) how to express the predicates conn_k on augmented trees using this more general logic $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A) \cup \Sigma)$.

4 Discussion

In this work we have added computational problems related to conn queries in graphs and the logic $\mathsf{FO}+\mathsf{conn}$ to the growing list of applications of the decomposition theorem of Cygan et al. [15]. Several questions can be asked about possible improvements of the obtained complexity bounds, especially regarding the data structure of Theorem 1.1.

First, in both the model-checking algorithm of Theorem 1.2 and the construction algorithm of the data structure of Theorem 1.1, the main bottleneck for the polynomial factor is the running time of the algorithm to compute the tree decomposition of Cygan et al. (see Theorem 2.1). This running time is cubic in the size of the graph, while in both applications presented in this work, the remainder of the construction takes fpt time with at most quadratic dependence on the graph size. Therefore, it seems imperative to revisit Theorem 2.1 with the purpose of improving the time complexity of the construction algorithm.

Second, in Section 6 of the full version we showed how to improve the time complexity of queries to polynomial in k by using the weakly unbreakable tree decomposition provided by [14]. The drawback is that the space usage of the data structure becomes $k^{\mathcal{O}(1)} \cdot |G|^2$. However, the quadratic dependence on $|G|$ is caused only by the brute-force implementation of the data structure for torso queries provided by Lemma 6.2 of the full version. So now we have two implementations of this data structure:

- Theorem 2.4 and Lemma 4.2 (of the full version) offers query time $q^{\mathcal{O}(1)}$ and space usage $2^{\mathcal{O}(q^2)} \cdot |T|$.
- Lemma 6.2 (of the full version) offers query time $q^{\mathcal{O}(1)}$ and space usage $q^{\mathcal{O}(1)} \cdot |T|^2$.

Is it possible to design a data structure that would offer query time $q^{\mathcal{O}(1)}$ while keeping the space usage at $q^{\mathcal{O}(1)} \cdot |T|$?

References

1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all fpt-equivalent. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1976–2004. SIAM, 2022.

2 Akanksha Agrawal, Lawqueen Kanesh, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. An FPT algorithm for elimination distance to bounded degree graphs. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021*, volume 187 of *LIPIcs*, pages 5:1–5:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 Akanksha Agrawal, Lawqueen Kanesh, Fahad Panolan, MS Ramanujan, and Saket Saurabh. An fpt algorithm for elimination distance to bounded degree graphs. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

4 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019*, pages 89–103. ACM, 2019.

5 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proceedings of the 15th International Conference on Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. doi:10.1007/11874683_11.

6 Mikołaj Bojańczyk. Separator logic and star-free expressions for graphs. *CoRR*, abs/2107.13953, 2021. arXiv:2107.13953.

7 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

8 Glencora Borradaile, Seth Pettie, and Christian Wulff-Nilsen. Connectivity oracles for planar graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 316–327. Springer, 2012.

9 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.

10 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017.

11 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

12 Thomas Colcombet. A combinatorial theorem for trees. In *Proceedings of the 34th International Colloquium Automata, Languages and Programming, ICALP 2007*, volume 4596 of *Lecture Notes in Computer Science*, pages 901–912. Springer, 2007. doi:10.1007/978-3-540-73420-8_77.

13 Bruno Courcelle. The Monadic Second-Order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

14 Marek Cygan, Paweł Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Transactions on Algorithms*, 17(1):6:1–6:30, 2021. doi:10.1145/3426738.

15 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum Bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019. doi:10.1137/140988553.

16 Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 465–474. ACM, 2010. doi:10.1145/1806689.1806754.

17 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM Journal on Computing*, 49(6):1363–1396, 2020. doi:10.1137/17M1146610.

18 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 121–131, 2014.

19 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

20 Fedor V Fomin, Petr A Golovach, Ignasi Sau, Giannos Stamoulis, and Dimitrios M Thilikos. A compound logic for modification problems: Big kingdoms fall from within. *arXiv preprint*, 2021. arXiv:2111.02755.

21 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Parameterized complexity of elimination distance to first-order logic properties. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021.

22 Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. In *Proceedings of the 5th Annual European Symposium on Algorithms, ESA 1997*, volume 1284 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1997.

23 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of mso1 model-checking. *Journal of Computer and System Sciences*, 80(1):180–194, 2014.

24 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

25 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

26 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In *47th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2021*, volume 12911 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2021. doi:10.1007/978-3-030-86838-3_6.

27 Bart MP Jansen, Jari JH de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 1757–1769, 2021.

28 Ken-ichi Kawarabayashi and Mikkel Thorup. The Minimum k-way Cut of bounded size is fixed-parameter tractable. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 160–169. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.53.

29 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.

30 Wojciech Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Log. Methods Comput. Sci.*, 16(1), 2020.

31 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010*, pages 189–198. IEEE Computer Society, 2010.

32 Alexander Lindermayr, Sebastian Siebertz, and Alexandre Vigny. Elimination distance to bounded degree on planar graphs. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPIcs*, pages 65:1–65:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

33 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractability of Graph Isomorphism in graphs with an excluded minor, 2021. Manuscript.

34 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

35 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for Min k-Cut. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 798–809. IEEE, 2020. doi:10.1109/FOCS46700.2020.00079.

36 Michał Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Toruńczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. *arXiv preprint v1*, 2021. arXiv:2111.03725.

37 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007*, pages 263–271. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.54.

38 Saket Saurabh and Meirav Zehavi. Parameterized complexity of multi-node hubs. In *Proceedings of the 13th International Symposium on Parameterized and Exact Computation, IPEC 2018*, volume 115 of *LIPIcs*, pages 8:1–8:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.IPEC.2018.8.

39 Nicole Schirrmacher, Sebastian Siebertz, and Alexandre Vigny. First-order logic with connectivity operators. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

40 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.

41 Szymon Toruńczyk. Aggregate queries on sparse databases. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020*, pages 427–443. ACM, 2020.

42 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.