Maintaining CMSO_2 Properties on Dynamic Structures with Bounded Feedback Vertex Number

Konrad Majewski
Institute of Informatics, University of Warsaw, Poland

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Marek Sokołowski
Institute of Informatics, University of Warsaw, Poland

Abstract

Let φ be a sentence of CMSO_2 (monadic second-order logic with quantification over edge subsets and counting modular predicates) over the signature of graphs. We present a dynamic data structure that for a given graph G that is updated by edge insertions and edge deletions, maintains whether φ is satisfied in G. The data structure is required to correctly report the outcome only when the feedback vertex number of G does not exceed a fixed constant k, otherwise it reports that the feedback vertex number is too large. With this assumption, we guarantee amortized update time $\mathcal{O}_{\varphi,k}(\log n)$.

By combining this result with a classic theorem of Erdős and Pósa, we give a fully dynamic data structure that maintains whether a graph contains a packing of k vertex-disjoint cycles with amortized update time $\mathcal{O}_k(\log n)$. Our data structure also works in a larger generality of relational structures over binary signatures.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases feedback vertex set, CMSO_2 formula, data structure, dynamic graphs, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.46

Related Version *Full Version*: `https://arxiv.org/abs/2107.06232` [13]

Funding This work 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).

1 Introduction

We consider data structures for graphs in a fully dynamic model, where the considered graph can be updated by the following operations: add an edge, remove an edge, add an isolated vertex, and remove an isolated vertex. Most of the contemporary work on data structures for graphs focuses on problems that in the static setting are polynomial-time solvable, such as connectivity or distance computation. In this work we follow a somewhat different direction and consider *parameterized problems*. That is, we consider problems that are NP-hard in the classic sense, even in the static setting, and we would like to design efficient dynamic data structures for them. The update time guarantees will typically depend on the size of the graph n and a parameter of interest k, and the goal is obtain as good dependence on n as possible while allowing exponential (or worse) dependence on k. The idea behind this approach is that the data structure will perform efficiently on instances where the parameter k is small, which is exactly the principle assumed in the field of parameterized complexity.

The systematic investigation of such *parameterized dynamic data structures* was initiated by Alman et al. [1], though a few earlier results of this kind can be found in the literature, e.g. [6, 7, 11]. Alman et al. revisited several techniques in parameterized complexity and developed their dynamic counterparts, thus giving suitable parameterized dynamic data structures for a number of classic problems, including Vertex Cover, Hitting Set, k-Path, and Feedback Vertex Set. The last example is important for our motivation. Recall that a *feedback vertex set* in an (undirected) graph G is a subset of vertices that intersects every cycle in G, and the *feedback vertex number* of G is the smallest size of a feedback vertex set in G. The data structure of Alman et al. monitors whether the feedback vertex number of a dynamic graph G is at most k (and reports a suitable witness, if so) with amortized update time $2^{\mathcal{O}(k \log k)} \cdot \log n$.

Dvořák et al. [6] and, more recently, Chen et al. [3] studied parameterized dynamic data structures for another graph parameter *treedepth*. Formally, the treedepth of a graph G is the least possible height of an *elimination forest* G: a rooted forest on the vertex set of G such that every edge of G connects a vertex with its ancestor. Intuitively, that a graph G has treedepth d means that G has a tree decomposition whose *height* is d, rather than width. Chen et al. [3] proved that in a dynamic graph of treedepth at most d, an optimum-height elimination forest can be maintained with update time $2^{\mathcal{O}(d^2)}$ (worst case, under the promise that the treedepth never exceeds d). This improved upon the earlier result of Dvořák et al. [6], who for the same problem achieved update time $f(d)$ for a non-elementary function f.

As already observed by Dvořák et al. [6], such a data structure can be used not only to the concrete problem of computing the treedepth, but more generally to maintaining satisfiability of any property that can be expressed in the *Monadic Second-Order logic* MSO_2. This logic extends standard First-Order logic FO by allowing quantification over subsets of vertices and subsets of edges, so it is able to express through constant-size sentences NP-hard problems such at Hamiltonicity or 3-colorability. More precisely, the following result was proved by Dvořák et al. [6] (see Chen et al. [3] for lifting the promise of boundedness of treedepth).

► **Theorem 1** ([3, 6]). *Given an MSO_2 sentence φ over the signature of graphs and $d \in \mathbb{N}$, one can construct a dynamic data structure that maintains whether a given dynamic graph G satisfies φ. The data structure is obliged to report a correct answer only when the treedepth of G does not exceed d, and otherwise it reports* Treedepth too large*. The updates work in amortized time $f(\varphi, d)$ for a computable function f, under the assumption that one is given access to a dictionary on the edges of G with constant-time operations.*

The proof of Theorem 1 is based on the following idea. If a graph G is supplied with an elimination forest of bounded depth, then, by the finite-state properties of MSO_2, whether φ is satisfied in G can be decided using a suitable bottom-up dynamic programming algorithm. Then it is shown that when G is updated by edge insertions and removals, one is able to maintain not only an optimum-height elimination forest F of G, but also a run of this dynamic programming algorithm on F. This blueprint brings the classic work on algorithmic meta-theorems in parameterized complexity to the setting of dynamic data structures, by showing that dynamic maintenance of a suitable decomposition is a first step to maintaining all properties that can be efficiently computed using this decomposition.

Notably, Chen et al. [3] apply this principle to two specific problems of interest: detection of k-paths and $(\geqslant k)$-cycles in undirected graphs. Using known connections between these objects and treedepth, they gave dynamic data structures for the detection problems that have update time $2^{\mathcal{O}(k^2)}$ for k-paths (assuming a dictionary on edges) and $2^{\mathcal{O}(k^4)} \cdot \log n$ for $(\geqslant k)$-cycles.

One of the main questions left open by the work of Dvořák et al. [6] and by Chen et al. [3] is whether in a dynamic graph of treewidth at most k it is possible to maintain a tree decomposition of width at most $f(k)$ with polylogarithmic update time. Note here that the setting of tree decompositions is the natural context in which MSO_2 on graphs is considered, due to Courcelle's Theorem [4], while the treedepth of a graph is always an upper bound on its treewidth. Thus, the works of Dvořák et al. [6] and of Chen et al. [3] can be regarded as partial progress towards resolving this question, where a weaker (larger) parameter treedepth is considered.

Our contribution. We approach the question presented above from another direction, by considering feedback vertex number – another parameter that upper bounds the treewidth. As mentioned, Alman et al. [1] have shown that there is a dynamic data structure that monitors whether the feedback vertex number is at most k with update time $2^{\mathcal{O}(k \log k)} \cdot \log n$. We extend this result by showing that in fact, every MSO_2-expressible property can be efficiently maintained in graphs of bounded feedback vertex number. Here is our main result.

▶ **Theorem 2.** *Given a sentence φ of CMSO_2 over the signature of graphs and $k \in \mathbb{N}$, one can construct a data structure that maintains whether a given dynamic graph G satisfies φ. The data structure is obliged to report a correct answer only if the feedback vertex number of G is at most k, otherwise it reports* Feedback vertex number too large*. The graph is initially empty and the amortized update time is $f(\varphi, k) \cdot \log n$, for some computable function f.*

Here, CMSO_2 is an extension of MSO_2 by modular counting predicates; this extends the generality slightly. Similarly as noted by Chen et al. [3], the appearance of the $\log n$ factor in the update time seems necessary: a data structure like the one in Theorem 2 could be easily used for connectivity queries in dynamic forests, for which there is an $\Omega(\log n)$ lower bound in the cell-probe model [15].

We prove Theorem 2 in a larger generality of relational structures over binary signatures, for a formal statement we refer the reader to the full version of the paper (Theorem 4.1). More precisely, we consider relational structures over signatures consisting of relation symbols of arity at most 2 that can be updated by adding and removing tuples from the relations, and by adding and removing isolated elements of the universe. In this language, graphs correspond to structures over a signature consisting of one binary relation signifying adjacency. As feedback vertex number we consider the feedback vertex number of the Gaifman graph of the structure. Generalization to relational structures is not just a mere extension of Theorem 2, it is actually a formulation that appears naturally in the inductive strategy that is employed in the proof.

As for this proof, we heavily rely on the approach used by Alman et al. [1] for monitoring the feedback vertex number. This approach is based on applying two types of simplifying operations, in alternation and a bounded number of times:

- contraction of subtrees in the graph; and
- removal of high-degree vertices.

We prove that in both cases, while performing the simplification it is possible to remember a bounded piece of information about each of the simplified parts, thus effectively enriching the whole data structure with information from which the satisfaction of φ can be inferred. Notably, for the contracted subtrees, this piece of information is the CMSO_2-type of appropriately high rank. To maintain these types in the dynamic setting, we use the top trees data structure of Alstrup et al. [2]. All in all, while our data structure is based on the same combinatorics of the feedback vertex number, it is by no means a straightforward lift of the work of Alman et al. [1]: enriching the data structure with information about types requires several new ideas and insights, both on the algorithmic and on the logical side of the reasoning. A more extensive discussion can be found in Section 2.

Applications. Similarly as in the work of Chen et al. [3], we observe that Theorem 2 can be used to obtain dynamic data structures for specific parameterized problems through a win/win approach. Consider the *cycle packing number* of a graph G: the maximum number of vertex-disjoint cycles that can be found in G. A classic theorem of Erdős and Pósa [9] states that there exists a universal constant c such that if the feedback vertex number of a graph G is larger than $c \cdot p \log p$, then the cycle packing number of G is at least p. We can use this result to establish the following.

▶ **Theorem 3.** *For a given $p \in \mathbb{N}$ one can construct a dynamic data structure that for a dynamic graph G (initially empty) maintains whether the cycle packing number of G is at least p. The amortized update time is $f(p) \cdot \log n$, for a computable function f.*

Proof. For a given p, it is straightforward to write a CMSO_2 sentence φ_p that holds in a graph G if and only if G contains p vertex-disjoint cycles. Then we may use the data structure of Theorem 2 for φ_p and $k = c \cdot p \log p$, where c is the constant given by the theorem of Erdős and Pósa [9]. Note that if this data structure reports that *Feedback vertex number too large*, then the cycle packing number is at least p, so this outcome can be reported. ◀

The same principle can be applied to other problems related to cycle packings and feedback vertex sets, e.g. Connected Feedback Vertex Set, Independent Feedback Vertex Set, and Tree Deletion Set. We discuss these applications in the full version of the paper (Section 7).

Organization. Due to space constraints, in this extended abstract we present only an overview of our approach with the intention of explaining the main conceptual points without going into technical details. Complete proofs can be found in the full version of this work, which is attached as the appendix.

2 Overview

In this section we present an overview of the proof of Theorem 2. We deliberately keep the description high-level in order to convey the main ideas. In particular, we focus on the graph setting and delegate the notation-heavy aspects of relational structures to the full exposition.

Let G be the given dynamic graph. We focus on the model where we have a promise that the feedback vertex number of G is at most k at all times. If we are able to construct a data structure in this promise model, then it is easy to lift this to the full model described in Theorem 2 using the standard technique of *postponing invariant-breaking insertions*. This technique was also used by Chen et al. [3] and dates back to the work of Eppstein et al. [8].

Colored graphs. We will be working with edge- and vertex-colored graphs. That is, if Σ is a finite set of colors (a *palette*), then a Σ-*colored graph* is a graph where every vertex and edge is assigned a color from Σ. In our case, all the palettes will be of size bounded by functions of k and the given formula φ, but throughout the reasoning we will use different (and rapidly growing) palettes. For readers familiar with relational structures, in general we work with relational structures over binary signatures (involving symbols of arity $0, 1, 2$), which are essentially colored graphs supplied with flags.

Thus, we assume that the maintained dynamic graph G is also a Σ-colored graph for some initial palette Σ. When G is updated by a vertex or edge insertion, we assume that the color of the new feature is provided with the update.

Monadic Second-Order Logic. Logic MSO_2 is Monadic Second-Order Logic with quantification over vertex subsets and edge subsets. This is a standard logic considered in parameterized complexity in connection with treewidth and Courcelle's Theorem. We refer to [5, Section 7.4] for a thorough introduction, and explain here only the main features. There are four types of variables: *individual* vertex/edge variables that evaluate to single vertices/edges, and *monadic* vertex/edge variables that evaluate to vertex/edge subsets. These can be quantified both existentially and universally. One can check equality of vertices/edges, incidence between an edge and a vertex, and membership of a vertex/edge to a vertex/edge subset. In case of colored graphs, one can also check colors of vertices/edges using unary predicates. Negation and all boolean connectives are allowed.

Note that in Theorem 2 we consider logic CMSO_2, which is an extension of the above by modular counting predicates that can be applied to monadic variables. For simplicity, we ignore this extension for the purpose of this overview.

Types. The key technical ingredient in our reasoning are *types*, which is a standard tool in model theory. We refer to the work of Grohe [10] for a more thorough introduction. Let G be a Σ-colored graph and q be a nonnegative integer. With G we can associate its *rank-q type* $\mathsf{tp}^q(G)$, which is a finite piece of data that contains all information about the satisfaction of MSO_2 sentences of quantifier rank at most q in G (i.e., with quantifier nesting bounded by q). More precisely:

- For every choice of q and Σ there is a finite set $\mathsf{Types}^{q,\Sigma}$ containing all possible rank-q types of Σ-colored graphs. The size of $\mathsf{Types}^{q,\Sigma}$ depends only on q and Σ.
- For every MSO_2 sentence ψ of quantifier rank at most q, the type $\mathsf{tp}^q(G)$ uniquely determines whether ψ holds in G.

In addition to the above, we also need an understanding that types are *compositional* under gluing of graphs along small boundaries. For this, we work with the notion of a *boundaried graph*, which is a graph G together with a specified subset of vertices ∂G, called the *boundary*. Typically, these boundaries will be of constant size. We extend the notion of a type to boundaried graphs, where the rank-q type $\mathsf{tp}^q(G)$ of a boundaried graph G contains information not only about all rank-q MSO_2 sentences satisfied in G, but also about all such sentences that in addition can use the vertices of ∂G as parameters (one can also think that vertices of ∂G are given through free variables). Again, for every finite set D, there is a finite set of possible types $\mathsf{Types}^{q,\Sigma}(D)$ of boundaried Σ-colored graphs with boundary D, and the size of $\mathsf{Types}^{q,\Sigma}(D)$ depends only on q, Σ, and $|D|$.

Now, on boundaried graphs there are two natural operations. First, if G is a boundaried graph and $u \in \partial G$, then one can *forget* u in G. This yields a boundaried graph $\mathsf{forget}(G, u)$ obtained from G by removing u from the boundary (otherwise the graph remains intact). Second, if G and H are two boundaried graphs and ξ is a partial bijection between ∂G and ∂H, then the *join* $G \oplus_\xi H$ is the boundaried graph obtained from the disjoint union of G and H by identifying vertices that correspond to each other in ξ; the new boundary is the union of the old boundaries (with identification applied).

With these notions in place, the compositionality of types can be phrased as follows:

- Given $\mathsf{tp}^q(G)$ and $u \in \partial G$, one can uniquely determine $\mathsf{tp}^q(\mathsf{forget}(G, u))$.
- Given $\mathsf{tp}^q(G)$ and $\mathsf{tp}^q(H)$ and a partial bijection ξ between the boundaries of G and H, one can uniquely determine $\mathsf{tp}^q(G \oplus_\xi H)$.

The determination described above is effective, that is, can be computed by an algorithm.

Top trees. We now move to the next key technical ingredient: the *top trees* data structure of Alstrup et al. [2]. Top trees work over a dynamic forest F, which is updated by edge insertions and deletions (subject to the promise that no update breaks acyclicity) and insertions and deletions of isolated vertices. For each connected component T of F one maintains a *top tree* Δ_T, which is a hierarchical decomposition of T into *clusters*. Each cluster S is a subtree of T with at least one edge that is assigned a boundary $\partial S \subseteq V(S)$ of size at most 2 with the following property: every vertex of S that has a neighbor outside of S belongs to ∂S. Formally, the top tree Δ_T is a binary tree whose nodes are assigned clusters in T so that:

- the root of Δ_T is assigned the cluster $(T, \partial T)$, where ∂T is a choice of at most two vertices in T;
- the leaves of Δ_T are assigned single-edge clusters;
- for every internal node x of Δ_T, the edge sets of clusters in the children of x form a partition of the edge set of the cluster at x.

Note that the last property implies that the cluster at x, treated as a boundaried graph, can be obtained from the two clusters at the children of x by applying the join operation, possibly followed by forgetting a subset of the boundary. We will then say that the cluster at x is obtained by *joining* the two clusters at its children.

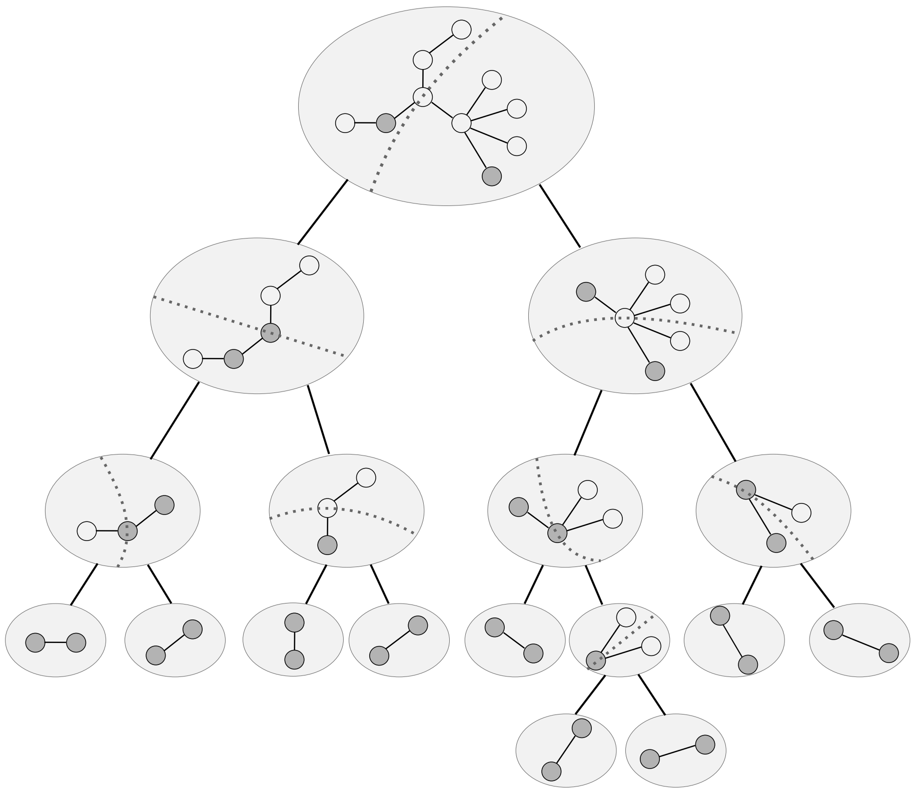

Figure 1 An example top tree Δ_T. Clusters correspond to light gray ovals. Boundary vertices in each cluster are marked dark gray. Note that in this example, Δ_T has two external boundary vertices. However, it may have fewer (zero or one) such vertices.

In [2], Alstrup et al. showed how to maintain, for a dynamic forest F, a forest of top trees $\{\Delta_T : T \text{ is a component of } F\}$ so that each tree Δ_T has depth $\mathcal{O}(\log n)$ and every operation is performed in worst-case time $\mathcal{O}(\log n)$. Moreover, they showed that the top trees data structure can be robustly enriched with various kinds of auxiliary information about clusters, provided this information can be efficiently composed upon joining clusters. More precisely, suppose that with each cluster C we can associate a piece of information $\mathcal{I}(C)$ so that

- $\mathcal{I}(C)$ can be computed in constant time when C has one edge; and
- if C is obtained by joining two clusters C_1 and C_2, then from $\mathcal{I}(C_1)$ and $\mathcal{I}(C_2)$ one can compute $\mathcal{I}(C)$ in constant time.

Then, as shown in [2], with each cluster C one can store the corresponding piece of information $\mathcal{I}(C)$, and still perform updates in time $\mathcal{O}(\log n)$.

In our applications, we work with top trees over dynamic Σ-colored forests, where with each cluster C we store information on its type:

$$\mathcal{I}(C) = \mathsf{tp}^p(C)$$

for a suitably chosen $p \in \mathbb{N}$. Here, for technical reasons we need to be careful about the colors: the type $\mathsf{tp}^p(C)$ takes into account the colors of all the edges of C and all the vertices of C *except* the vertices of ∂C (formally, we consider the type of C with colors stripped from boundary vertices). The rationale behind this choice is that a single vertex u can participate in the boundary of multiple clusters, hence in the dynamic setting we cannot afford to update the type of each of them upon updating the color of u. Rather, every cluster C stores its type with the colors on ∂C stripped, and if we wish to compute the type of C with these colors included, it suffices to look up those colors and update the stripped type (using compositionality).

Brushing these technical details aside, after choosing the definitions right, the compositionality of types explained before perfectly fits the properties required from an enrichment of top trees. This means that with each cluster C we can store $\mathsf{tp}^p(C)$ while guaranteeing worst-case update time $\mathcal{O}_{p,\Sigma}(\log n)$. We remark that the combination of top trees and MSO_2 types appears to be a novel contribution of this work; we hope that it can be reused in the future.

So if F is a dynamic Σ-colored forest and p is a parameter, then for each tree T in F we can maintain a top tree Δ_T whose root is supplied with the type $\mathsf{tp}^p(T)$. Knowing the multiset of rank-p types of trees in F, we can use standard compositionality and idempotence of types to compute the type $\mathsf{tp}^p(F)$, from which in turn one can infer which rank-p sentences are satisfied in F. By taking p to be the quantifier rank of a given sentence φ, we obtain:

▶ **Theorem 4.** *Let Σ be a finite palette and φ be an MSO_2 sentence over Σ-colored graphs. Then there is a dynamic data structure that for a dynamic Σ-colored forest F maintains whether φ holds in F. The worst-case update time is $\mathcal{O}_{\varphi,\Sigma}(\log n)$.*

Note that the statement of Theorem 4 matches (the colored version of) the statement of Theorem 2 for $k = 0$. Curiously, we are not aware of this result existing already in the literature, despite the naturality of the problem. We remark that maintaining MSO queries over dynamic forests has been considered in the databases literature, see [14] and references therein, however under a different (and somewhat orthogonal) set of allowed updates.

The data structure of Alman et al. [1]. Our goal now is to lift Theorem 4 to the case of $k > 0$. For this we rely on the approach of Alman et al. [1] for monitoring the feedback vertex number, which is based on a sparsity-based strategy that is standard in parameterized complexity, see e.g. [5, Section 3.3].

The approach is based on two lemmas. The first one concerns the situation when the graph contains a vertex u of degree at most 2. In this case, it is safe to dissolve u: either remove it, in case it has degree 0 or 1, or replace it with a new edge connecting its neighbors, in case it has degree 2. Note that dissolving a degree-2 vertex naturally can create a multigraph. This creates technical issues both in [1] and in this work, but we shall largely ignore them for the purpose of this overview. Formally, we have the following.

▶ **Lemma 5** (folklore). *Dissolving a vertex of degree at most* 2 *in a multigraph does not change the feedback vertex number.*

The second lemma concerns the situation when the graph has minimum degree at least 3. Then a sparsity-based argument shows that every feedback vertex set of size at most k intersects the set of $\mathcal{O}(k)$ vertices with highest degrees.

▶ **Lemma 6** (Lemma 3.3 in [5]). *Let G be a multigraph with minimum degree* 3 *and let B be the set of $3k$ vertices with highest degrees in G. Then every feedback vertex set of size at most k in G intersects B.*

Lemmas 5 and 6 can be used to obtain an fpt algorithm for Feedback Vertex Set with running time $(3k)^k \cdot (n+m)$ (see [5, Theorem 3.5]): apply the reduction of Lemma 5 exhaustively, and then branch on which of the $3k$ vertices with highest degrees should be included in the solution. This results in a recursion tree of total size at most $(3k)^k$.

The data structure of Alman et al. [1] is based on dynamization of the branching algorithm presented above. There are two main challenges:

- dynamic maintenance of the sequence of dissolutions given by Lemma 5; and
- dynamic maintenance of the set of high degree vertices.

For the first issue, it is explanatory to imagine performing the dissolutions not one by one iteratively, but all at once. It is not hard to see that the result of applying Lemma 5 exhaustively is that the input multigraph G gets contracted to a multigraph $\mathsf{Contract}(G)$ in the following way: the edge set of G into disjoint trees, and each of them either disappears or is contracted into a single edge in $\mathsf{Contract}(G)$; see Figure 2 for a visualization. (There may be some corner cases connected to loops in $\mathsf{Contract}(G)$ that result from contracting not trees, but unicyclic graphs; we ignore this issue in this overview.) We call the elements of this partition *ferns*, and the corresponding decomposition of G into ferns is called the *fern decomposition* of G. Importantly, the order of performing the contractions has no effect on the outcome, yielding always the same fern decomposition of G.

With each fern of S we can associate its boundary ∂S, which is the set of vertices of S incident to edges that lie outside of S. It is not hard to see that this boundary will always be of size 0, 1, or 2. The ferns that correspond to edges in $\mathsf{Contract}(G)$ are the ferns with boundary of size 2 (each such fern gets contracted to an edge connecting the two vertices of the boundary) and non-tree ferns with boundary of size 1 (each such fern gets contracted to a loop at the unique vertex of the boundary).

The idea of Alman et al. is to maintain the ferns in the fern decomposition using link-cut trees. It is shown that each update in G affects the fern decomposition only slightly, in the sense that it can be updated using a constant number of operations on link-cut trees. In this way, the fern decomposition and the graph $\mathsf{Contract}(G)$ can be maintained with worst-case $\mathcal{O}(\log n)$ time per update in G. This resolves the first challenge.

For the second challenge, Alman et al. observe that if in Lemma 6 one increases the number of highest degree vertices included in B from $3k$ to $12k$, then the set remains "valid" – in the sense of satisfying the conclusion of the lemma – even after $\mathcal{O}(m/k)$ updates are applied to the graph. Here, m denotes the number of edges of the graph on which Lemma 6 is applied, which is $\mathsf{Contract}(G)$ in our case. This means that it remains correct to perform a recomputation of the set B only every $\Theta(m/k)$ updates. Since such a recomputation takes time $\mathcal{O}(m)$, the amortized update time is $\mathcal{O}(k)$.

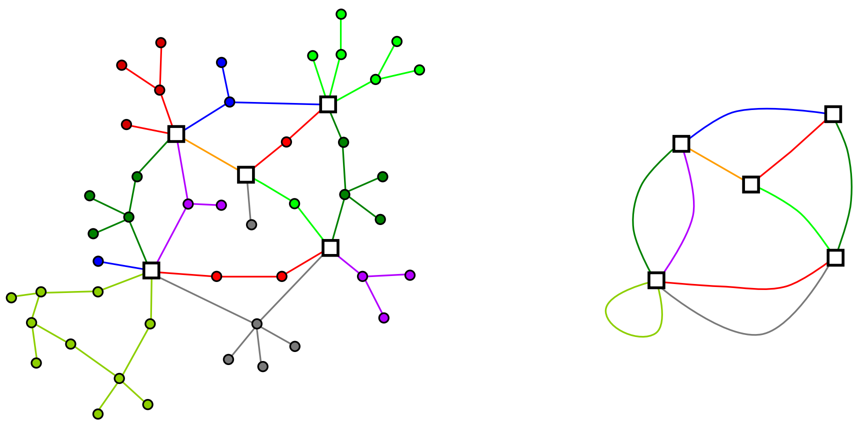

Figure 2 Left: A graph G together with its fern decomposition. Different ferns are depicted with different colors; these should not be confused with the coloring of edges of G with colors from Σ. Right: The multigraph $\mathsf{Contract}(G)$ obtained by contracting each fern. Note that in the construction of $\mathsf{Contract}^p(G)$ described in the discussion of the Contraction Lemma, we would not have parallel edges or loops. Instead, each pack of parallel edges would be replaced by a single one, colored with the joint type of the whole pack. Similarly, loops on a vertex would be removed and their joint type would be stored in the color of the vertex.

Once $\mathsf{Contract}(G)$ and $B \subseteq V(\mathsf{Contract}(G))$ are known, Lemma 6 asserts that if the feedback vertex number of G is at most k, there exists a vertex $b \in B$ whose deletion decreases the feedback vertex number. Therefore, the idea of Alman et al. is to construct a recursive copy of the data structure for each $b \in B$: the copy maintains the graph $\mathsf{Contract}(G) - b$ and uses parameter $k-1$ instead of k. Note that when B gets recomputed, all these data structures are reset, but thanks to amortization we have time to do it.

All in all, once one unravels the recursion, the whole construction is a tree of data structures of depth k and branching $12k$, which is maintained with amortized update time $2^{\mathcal{O}(k \log k)} \cdot \log n$. The graph has feedback vertex number at most k if and only if this tree contains at least one leaf with an empty graph.

Our data structure. We now describe the high-level idea of our data structure.

Lemmas 5 and 6 can be used not only to design an fpt algorithm for Feedback Vertex Set, but also an approximation algorithm. Consider the following procedure: apply the reduction of Lemma 5 exhaustively, then greedily take *all* the $3k$ vertices with highest degrees to the constructed feedback vertex set, and iterate these two steps in alternation until the graph becomes empty. Lemma 6 guarantees that provided the feedback vertex number was at most k in the first place, the iteration terminates after at most k steps; the $3k^2$ selected vertices form a feedback vertex set. We note that this application of Lemmas 5 and 6 for feedback vertex set approximation is not new, for instance it was recently used by Kammer and Sajenko [12] in the context of space-efficient kernelization.

Our data structure follows the design outlined above. That is, instead of a tree of data structures, we maintain a sequence of $2k+2$ data structures, respectively for multigraphs

$$G_0, H_0, G_1, H_1, \ldots, G_k, H_k.$$

These multigraphs essentially satisfy the following:

- $G_0 = G$;
- $H_i = \mathsf{Contract}(G_i)$ for $i = 0, 1, \ldots, k$; and
- $G_{i+1} = G_i - B_i$ for $i = 0, 1, \ldots, k-1$, where B_i is a set that satisfies the conclusion of Lemma 6 for G_i.

Note that these invariants imply that provided the feedback vertex number of G is at most k, the feedback vertex number of G_i and of H_i is at most $k-i$ for each $i \in \{0,1,\ldots,k\}$, implying that G_k is a forest and H_k is the empty graph.

The precise definitions of $\mathsf{Contract}(\cdot)$ and of deleting vertices used in the sequence above will be specified later. More precisely, graphs $G_0, H_0, \ldots, G_k, H_k$ will be colored with palettes $\Sigma_0, \Gamma_0, \ldots, \Sigma_k, \Gamma_k$ in order, where $\Sigma_0 = \Sigma$. These palettes will grow (quite rapidly) in sizes, but each will be always of size bounded in terms of k, Σ, and q – the quantifier rank of the fixed sentence φ whose satisfaction we monitor. The idea is that when obtaining H_i from G_i by contracting ferns, we use colors from Γ_i to store information about the contracted ferns on edges and vertices of H_i. Similarly, when removing vertices of B_i from H_i to obtain G_{i+1}, we use colors from Σ_{i+1} on vertices of G_{i+1} to store information about the adjacencies of the removed vertices. These steps are encompassed by two key technical statements – the Contraction Lemma and the Downgrade Lemma – which we explain below.

Contraction Lemma. We explain the Contraction Lemma for the construction of $H \coloneqq H_0$ from $G = G_0$; the construction for $i > 0$ is the same. Recall that eventually we are interested in monitoring whether the given sentence φ is satisfied in G. For this, it is sufficient to monitor the type $\mathsf{tp}^q(G)$, where q is the quantifier rank of φ. Consider the following construction:

- Pick some large $p \in \mathbb{N}$.
- Consider the fern decomposition $\mathcal{F}$ of G and let $\mathcal{K} \coloneqq \{\partial S \colon S \in \mathcal{F}\}$. For every $D \in \mathcal{K}$, let R_D be the join of all the ferns with boundary D, and with colors stripped from the vertices of D. Note that R_D is a boundaried graph with boundary D.
- For every $D \in \mathcal{K}$ with $|D| = 2$, contract R_D to a single edge with color $\mathsf{tp}^p(R_D)$ connecting the two vertices of D.
- For every $D \in \mathcal{K}$ with $|D| = 1$, contract R_D onto the single vertex d of D, and make d of color $\mathsf{tp}^p(R_D)$.
- Remove $R_\emptyset$, if present, and remember $\mathsf{tp}^p(R_\emptyset)$ through flags[1].
- The obtained colored graph is named $\mathsf{Contract}^p(G)$. Note that $\mathsf{Contract}^p(G)$ is a Γ^p-colored graph, where Γ^p is a palette consisting of all rank-p types of Σ-colored graphs with a boundary of size at most 2.

Thus, every fern S in G is essentially disposed of, but a finite piece of information (the rank-p type) about S is being remembered in $\mathsf{Contract}^p(G)$ on the boundary of S. The intuition is that if p is large enough, these pieces of information are enough to infer the rank-q type of G. This intuition is confirmed by the following Replacement Lemma.

▶ **Lemma 7** (Replacement Lemma, informal statement)**.** *For any given $q \in \mathbb{N}$ and Σ, there exists $p \in \mathbb{N}$ large enough so that for any Σ-colored graph G, the type $\mathsf{tp}^p(\mathsf{Contract}^p(G))$ uniquely determines the type $\mathsf{tp}^q(G)$.*

The proof of the Replacement Lemma uses Ehrenfeucht-Fraïsse games. It is conceptually rather standard, but technically quite involved. We note that the obtained constant p is essentially the number of rank-q types of Σ-colored graphs, which is approximately a tower of exponentials of height q applied to $|\Sigma|$. Since Replacement Lemma is used k times in the construction, this incurs a huge explosion in the parameter dependence in our data structure.

Replacement Lemma shows that in order to monitor the type $\mathsf{tp}^q(G)$ in the dynamic setting, it suffices to maintain the graph $H \coloneqq \mathsf{Contract}^p(G)$ and the type $\mathsf{tp}^p(H)$. Maintaining H dynamically is the responsibility of the Contraction Lemma.

[1] We assume that a colored graph can be supplied with a bounded number of boolean flags, which thus can store a bounded amount of additional information. In the general setting of relational structures, flags are modeled by nullary predicates (predicates of arity 0).

▶ **Lemma 8** (Contraction Lemma, informal statement)**.** *For a given $p \in \mathbb{N}$ and palette Σ, there is a dynamic data structure that for a dynamic graph G, maintains the graph* $\mathsf{Contract}^p(G)$ *under updates in G. The worst-case update time is $\mathcal{O}_{p,\Sigma}(\log n)$.*

The proof of Lemma 8 follows closely the reasoning of Alman et al. [1]. That is, in the same way as in [1], every update in G incurs a constant number of changes in the fern decomposition of G, expressed as splitting or merging of individual ferns. Instead of relying on link-cut trees as in [1], the ferns are stored using top trees. This is because we enrich the top trees data structure with the information about rank-p types of clusters, as in Theorem 4, so that for each fern S we know its rank-p type. This type is needed to determine the color of the feature (edge/vertex/flag) in $H = \mathsf{Contract}^p(G)$ to which S contributes.

Executing the plan sketched above requires an extreme care about details. Note for instance that in the construction of $\mathsf{Contract}^p(G)$, when defining R_D we explicitly stripped colors from the boundary vertices. This is for a reason similar to that discussed alongside Theorem 4: including the information on the colors of D in $\mathsf{tp}^p(R_D)$ would mean that a single update to the color of a vertex d would affect the types of all subgraphs R_D with $d \in D$, and there is potentially an unbounded number of such subgraphs. Further, we remark that Alman et al. [1] relied on an understanding of the fern decomposition through a sequence of dissolutions, which makes some arguments inconvenient for generalization to our setting. We need a firmer grasp on the notion of fern decomposition, hence we introduce a robust graph-theoretic description that is static – it does not rely on an iterative dissolution procedure. This robustness helps us greatly in maintaining ferns and their types in the dynamic setting.

Another noteworthy technical detail is that the operator $\mathsf{Contract}^p(\cdot)$, as defined above, does not create parallel edges or loops, and thus we stay within the realm of colored simple graphs (or, in the general setting, of classic relational structures over binary signatures). Unfortunately, this simplification cannot be applied throughout the whole proof, as in Lemma 6 we need to count the degrees with respect to the multigraph $\mathsf{Contract}(G)$ as defined in Alman et al. [1]. For this reason, in the full proof we keep trace of two objects at the same time: a relational structure $\mathbb{A}$ that we are interested in, and a multigraph H which is a supergraph of the Gaifman graph of $\mathbb{A}$ and that represents the structure of earlier contractions.

Downgrade Lemma. Finally, we are left with the Downgrade Lemma, which is responsible for the reducing the graph by removing a bounded number of vertices. Formally, we have a Γ-colored graph H and a set B of $\mathcal{O}(k)$ vertices, and we would like to construct a Σ'-colored graph $G' = \mathsf{Downgrade}(H, B)$ by removing the vertices of B and remembering information about them on the remaining vertices of H. This construction is executed as follows:

- Enumerate the vertices of B as $b_1, \ldots, b_\ell$, where $\ell = |B|$.
- Construct G' by removing vertices of B.
- For every color $c \in \Gamma$ and $i \in \{1, \ldots, \ell\}$, add to G' a flag signifying whether b_i has color c in G.
- For every pair $i, j \in \{1, \ldots, \ell\}$, $i < j$, and every color $c \in \Gamma$ add to G' a flag signifying whether b_i and b_j are connected in G by an edge of color c.
- For every vertex $u \in V(G) \setminus B$, every $i \in \{1, \ldots, \ell\}$, and every color $c \in \Gamma$, refine the color of u in G' by adding the information on whether u and b_i were connected in G by an edge of color c.
- The obtained graph is the graph G'. Note that G' is Σ'-colored, where $\Sigma' = \Gamma \times 2^{[\ell] \times \Gamma}$.

Thus, the information about vertices of B and edges incident to B is being stored in flags and colors on vertices of $V(G) \setminus B$. We have the following analogue of the Replacement Lemma.

▶ **Lemma 9.** *For any given $p \in \mathbb{N}$, there exists $q' \in \mathbb{N}$ large enough so that for any Γ-colored graph H and a subset B of $\mathcal{O}(k)$ vertices, the type $\mathsf{tp}^{q'}(\mathsf{Downgrade}(H, B))$ uniquely determines $\mathsf{tp}^{p}(H)$.*

The proof of Lemma 9 is actually very simple and boils down to a syntactic modification of formulas. From Lemma 9 it follows that to maintain the type $\mathsf{tp}^{p}(H)$, it suffices to maintain a bounded-size set B satisfying the conclusion of Lemma 6, the graph $G' = \mathsf{Downgrade}(H, B)$, and its type $\mathsf{tp}^{q'}(G')$. This is the responsibility of the Downgrade Lemma.

▶ **Lemma 10** (Downgrade Lemma, informal statement)**.** *For a given $p \in \mathbb{N}$ and palette Γ, there is a dynamic data structure that for a dynamic graph H of feedback vertex number at most k and with minimum degree 3, maintains a set of vertices $B \subseteq V(H)$ with $|B| \leqslant 12k$ and satisfying the conclusion of Lemma 6, and the graph $\mathsf{Downgrade}(H, B)$. The amortized update time is $\mathcal{O}_{p,\Gamma,k}(\log n)$.*

The proof of the Downgrade Lemma is essentially the same as that given for the corresponding step in Alman et al. [1]. We recompute B from scratch every $\Theta(m/k)$ updates, because the argument of Alman et al. shows that B remains valid for this long. Recomputing B implies recomputing $\mathsf{Downgrade}(H, B)$ in $\mathcal{O}_{p,\Gamma,k}(m)$ time, so the amortized complexity is $\mathcal{O}_{p,\Gamma,k}(1)$ (there are additional logarithmic factors from auxiliary data structures).

Endgame. We now have all the pieces to assemble the proof of Theorem 2. Let q_0 be the quantifier rank of the given sentence φ and let $G_0 = G$ be the considered dynamic graph. By Replacement Lemma, to monitor $\mathsf{tp}^{q_0}(G_0)$ (from which the satisfaction of φ can be inferred), it suffices to monitor $\mathsf{tp}^{p_0}(H_0)$, where $H_0 \coloneqq \mathsf{Contract}^{p_0}(G_0)$ and p_0 is as provided by the Replacement Lemma. By Contraction Lemma, we can efficiently maintain H_0 under updates of G_0. By Lemma 9, to monitor $\mathsf{tp}^{p_0}(H_0)$ it suffices to monitor $\mathsf{tp}^{q_1}(G_1)$, where $G_1 \coloneqq \mathsf{Downgrade}(H_0, B_0)$, and B_0 is a set that satisfies the conclusion of Lemma 6. By Downgrade Lemma, we can efficiently maintain such a set B_0 and the graph G_1. We proceed further in this way, alternating the usage of the Contraction Lemma and the Downgrade Lemma. Observe that each application of Downgrade Lemma strictly decrements the feedback vertex number, so after k steps we end up with an empty graph H_k. The type of this graph can be directly computed from its flags, and this type can be translated back to infer $\mathsf{tp}^{q}(G)$ by using Replacement Lemma and Lemma 9 alternately.

References

1 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic Parameterized Problems and Algorithms. *ACM Trans. Algorithms*, 16(4):45:1–45:46, 2020. doi:10.1145/3395037.

2 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

3 Jiehua Chen, Wojciech Czerwinski, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartlomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

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

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22nd Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.

7 Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Symposium on Algorithms and Data Structures, WADS 2013*, volume 8037 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2013. doi:10.1007/978-3-642-40104-6_27.

8 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

9 Paul Erdős and Lajos Pósa. On the maximal number of disjoint circuits of a graph. *Publ. Math. Debrecen*, 9:3–12, 1962.

10 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

11 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 241–252. Springer, 2014. doi:10.1007/978-3-319-08404-6_21.

12 Frank Kammer and Andrej Sajenko. FPT-space graph kernelizations. *CoRR*, abs/2007.11643, 2020. arXiv:2007.11643.

13 Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Maintaining CMSO_2 properties on dynamic structures with bounded feedback vertex number. *CoRR*, abs/2107.06232, 2021. arXiv:2107.06232.

14 Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 769–778. ACM, 2018. doi:10.1145/3209108.3209144.

15 Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 546–553. ACM, 2004.