Query Maintenance Under Batch Changes with Small-Depth Circuits

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, Chennai, India

Asif Khan
Chennai Mathematical Institute, India

Anish Mukherjee
University of Warwick, Coventry, UK

Felix Tschirbs
Ruhr University Bochum, Germany

Nils Vortmeier
Ruhr University Bochum, Germany

Thomas Zeume
Ruhr University Bochum, Germany

Abstract

Which dynamic queries can be maintained efficiently? For constant-size changes, it is known that constant-depth circuits or, equivalently, first-order updates suffice for maintaining many important queries, among them reachability, tree isomorphism, and the word problem for context-free languages. In other words, these queries are in the dynamic complexity class DynFO. We show that most of the existing results for constant-size changes can be recovered for batch changes of polylogarithmic size if one allows circuits of depth $\mathcal{O}(\log \log n)$ or, equivalently, first-order updates that are iterated $\mathcal{O}(\log \log n)$ times.

2012 ACM Subject Classification Theory of computation → Logic and databases; Theory of computation → Complexity theory and logic

Keywords and phrases Dynamic complexity theory, parallel computation, dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.46

Related Version *Full Version*: https://arxiv.org/abs/2407.20031

Funding *Samir Datta*: Partially funded by a grant from Infosys foundation.
Asif Khan: Partially funded by a grant from Infosys foundation.
Anish Mukherjee: Research was supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP) and by EPSRC award EP/V01305X/1.
Felix Tschirbs: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.
Nils Vortmeier: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.
Thomas Zeume: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.

1 Introduction

Dynamic descriptive complexity [23, 10] is a framework for studying the amount of resources that are necessary to *maintain* the result of a query when the input changes slightly, possibly using additional auxiliary data (which needs to be maintained as well). Its main class DynFO contains all queries for which the update of the query result (and possibly of further useful

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 46; pp. 46:1–46:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

auxiliary data) can be expressed in first-order logic FO. Equivalently[1], the updates can be computed using (DLOGTIME) uniform circuits with constant-depth and polynomial size that consist of $\neg$- as well as $\wedge$- and $\vee$-gates with unbounded fan-in, that is, within uniform AC^0.

It is known that many important queries can be maintained in DynFO if only one bit of the input changes in every step. This includes reachability for acyclic graphs [11, 23], undirected graphs [23, 12, 17], and general directed graphs [6], tree isomorphism [14] and every problem definable in monadic second-order logic MSO for graphs of bounded treewidth [8], all under insertions and deletions of single edges. Also, membership in context-free languages can be maintained under changes of single positions of the input word [15].

Some of these results have been extended to changes beyond single-bit changes: reachability in undirected graphs is in DynFO if simultaneously $\text{polylog}(n) = (\log n)^{\mathcal{O}(1)}$ edges can be inserted or deleted [7], where n is the size of the graph; regular languages are in DynFO under changes of $\text{polylog}(n)$ positions at once [24]. Reachability in directed graphs can be maintained under insertions and deletions of $\mathcal{O}(\frac{\log n}{\log \log n})$ many edges [9].

Thus, only for few problems it is known that changes of polylogarithmic size (or: even non-constant size) can be handled in DynFO, or, equivalently, by AC^0-updates. Trivially, if a problem can be maintained in DynFO under single-bit changes it can also be maintained under $\text{polylog}(n)$ changes using AC-circuits of $\text{polylog}(n)$-depth. This is achieved by processing the changed bits "sequentially" by "stacking" $\text{polylog}(n)$ copies of the constant-depth circuit for processing single-bit changes.

The starting point for the present paper is the question which problems can be maintained by AC-circuits of less than $\text{polylog}(n)$ depth under $\text{polylog}(n)$-sized changes, in particular which of the problems known to be in DynFO under single-bit changes. The answer is short: for almost all of them circuits of depth $\mathcal{O}(\log \log n)$ suffice.

A first observation is that directed reachability under polylogarithmic changes can be maintained by AC-circuits of depth $\mathcal{O}(\log \log n)$. This can be derived by analyzing the proof from [9] (see Section 3). For this reason, we introduce the dynamic complexity class[2] DynFOLL of problems that can be maintained using circuits with polynomial size and depth $\mathcal{O}(\log \log n)$ or, equivalently, by first-order formulas that are iterated $\mathcal{O}(\log \log n)$ times. We investigate its power when changes affect $\text{polylog}(n)$ input bits and prove that almost all problems known to be maintainable in DynFO for constant-size changes fall into this class for changes of $\text{polylog}(n)$-size, see Table 1. One important problem left open is whether all MSO-definable queries for bounded treewidth graphs can be maintained in DynFOLL under $\text{polylog}(n)$ changes. We present an intermediate result and show that tree decompositions can be maintained within DynFOLL (see Section 5).

This power of depth-$\mathcal{O}(\log \log n)$ update circuits came as a surprise to us. Statically, circuits of this depth and polynomial size still cannot compute the parity of n bits due to Håstad's famous lower bound for parity: depth-$(d+1)$ AC-circuits with alternating $\wedge$- and $\vee$-layers require $2^{\Omega(n^{1/d})}$ gates for computing parity (see, e.g., [20, Theorem 12.3]). Dynamically, while such update circuits are powerful for changes of non-constant size, they seem to provide not much more power for single-bit changes. As an example, the parity-exists query from [26] is conjectured to not be in DynFO, and it also cannot easily be seen to be in DynFOLL.

---

[1] assuming that first-order formulas have access to numeric predicates $\leq, +, \times$

[2] The class could equally well be called (uniform) $\mathsf{DynAC}[\log \log n]$. We opted for the name DynFOLL as it extends DynFO and its static variant was introduced as FOLL [2].

Table 1 Overview of results for DynFO and DynFOLL. Entries indicate the size of changes that can be handled by DynFO and DynFOLL programs, respectively.

Dynamic query	DynFO		DynFOLL	
reachability				
general graphs	$\mathcal{O}(\frac{\log n}{\log\log n})$	[9]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 3)
undirected graphs	$(\log n)^{\mathcal{O}(1)}$	[7]	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 4)
acyclic graphs	$\mathcal{O}(\frac{\log n}{\log\log n})$	[9]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 3)
distances				
general graphs	open		open	
undirected graphs	$\mathcal{O}(1)$	[17]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 6)
acyclic graphs	$\mathcal{O}(1)$		$(\log n)^{\mathcal{O}(1)}$	(Theorem 6)
bounded tree width				
tree decomposition	open		$(\log n)^{\mathcal{O}(1)}$	(Theorem 16)
MSO properties	$\mathcal{O}(1)$	[8]	$\mathcal{O}(1)$	
other graph problems				
tree isomorphism	$\mathcal{O}(1)$	[14]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 12)
minimum spanning forest	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 5)
maximal matching	$\mathcal{O}(1)$	[23]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)
$(\delta + 1)$-colouring	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)	n^2	(static, [16])
word problem				
regular languages	$(\log n)^{\mathcal{O}(1)}$	[24]	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 4)
context-free languages	$\mathcal{O}(1)$	[15]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 9)

The obtained bounds are almost optimal. For all mentioned problems, DynFO can handle changes of size at most polylog(n) and DynFOLL can handle changes of size at most $(\log n)^{\mathcal{O}(\log\log n)}$. This is an immediate consequence of Håstad's lower bound for parity and standard reductions from parity to these problems. For the queries that are known to be maintainable under polylog(n) changes in DynFO, we show that they can be maintained under $(\log n)^{\mathcal{O}(\log\log n)}$ changes in DynFOLL.

Our results rely on two main techniques for handling changes of polylogarithmic size:

- In the *small-structure technique* (see Section 3), it is exploited that on structures of polylogarithmic size, depth-$\mathcal{O}(\log\log n)$ circuits have the power of NC^2 circuits. Dynamic programs that use this technique first construct a substructure of polylogarithmic size depending on the changes and the current auxiliary data, then perform a NC^2-computation on this structure, and finally combine the result with the rest of the current auxiliary data to obtain the new auxiliary data. This technique is a slight generalization of previously used techniques for DynFO.
- In the *hierarchical technique* (see Section 4), it is exploited that auxiliary data used in dynamic programs is often "composable". Dynamic programs that use this technique first construct polynomially many structures depending on the current auxiliary data, each of them associated with one of the changes (in some cases, known dynamic programs for single changes can be exploited for this step). Then, in $\mathcal{O}(\log\log n)$ rounds, structures are combined hierarchically such that after ℓ rounds the program has computed polynomially many structures, each associated with 2^ℓ changes.

2 Preliminaries and setting

We introduce some notions of finite model theory, circuit complexity and the dynamic complexity framework.

Finite model theory & circuit complexity. A (relational) schema σ is a set of relation symbols and constant symbols. A relational structure $\mathcal{S}$ over a schema σ consists of a finite domain D, relations $R^{\mathcal{S}} \subseteq D^k$ for every k-ary relation symbol $R \in \sigma$, and interpretations $c^{\mathcal{S}} \in D$ of every constant symbol $c \in \sigma$. We assume in this work that every structure has a linear order $\leq$ on its domain. We can therefore identify D with the set $\{0, \dots, n-1\}$.

First-order logic FO is defined in the usual way. Following [19], we allow first-order formulas to access the linear order on the structures and corresponding relations $+$ and $\times$ encoding addition and multiplication. We write $\mathsf{FO}(\leq, +, \times)$ to make this explicit. $\mathsf{FO}(\leq, +, \times)$ can express iterated addition and iterated multiplication for polylogarithmically many numbers that consist of polylog(n) bits, see [18, Theorem 5.1].

First-order logic with $\leq, +, \times$ is equivalent to (DLOGTIME) uniform AC^0, the class of problems decidable by uniform families of constant-depth circuits with polynomially many "not"-, "and"- and "or"-gates with unbounded fan-in. We write $\mathsf{AC}[f(n)]$ for the class that allows for polynomial-sized circuits of depth $\mathcal{O}(f(n))$, where n is the number of input bits. For polynomially bounded and first-order constructible functions f, the class $\mathsf{AC}[f(n)]$ is equal to $\mathsf{IND}[f(n)]$, the class of problems that can be expressed by inductively applying an $\mathsf{FO}(\leq, +, \times)$ formula $\mathcal{O}(f(n))$ times [19, Theorem 5.22]. So, we can think of an $\mathsf{AC}[f(n)]$ circuit as being a stack of $\mathcal{O}(f(n))$ copies of some AC^0 circuit. The class FOLL, see [2], is defined as $\mathsf{IND}[\log\log n] = \mathsf{AC}[\log\log n]$.

The circuit complexity classes uniform NC^i and SAC^i are defined via uniform circuits of polynomial size and depth $\mathcal{O}((\log n)^i)$; besides "not"-gates, NC circuits use "and"- and "or"-gates with fan-in 2, SAC circuits allow for "or"-gates with unbounded fan-in.

Dynamic complexity. The goal of a *dynamic program* Π is to maintain the result of a query applied to an input structure $\mathcal{I}$ that is subject to changes. In this paper, we consider changes of the form $\textsc{ins}_R(P)$, the insertion of a set P of tuples into the relation R of $\mathcal{I}$, and $\textsc{del}_R(P)$, the deletion of the set P from R. We usually restrict the size of the set P to be bounded by a function $s(n)$, where n is the size of the domain of $\mathcal{I}$. Most of the time, the bound is polylogarithmic in n, so $s(n) = \log(n)^c$ for some constant c. A pair (Q, Δ) of a query Q and a set Δ of (size-bounded) change operation $\textsc{ins}_R, \textsc{del}_R$ is called a *dynamic query*.

To maintain some dynamic query over σ-structures, for some schema σ, Π stores and updates a set $\mathcal{A}$ of auxiliary relations over some schema σ_{aux} and over the same domain as the input structure. For every auxiliary relation symbol $A \in \sigma_{\text{aux}}$ and every change operation δ, Π has an update program $\varphi_\delta^A(\bar{x})$, which can access input and auxiliary relations. Whenever an input structure $\mathcal{I}$ is changed by a change $\delta(P)$, resulting in the structure $\mathcal{I}'$, the new auxiliary relation $A^{\mathcal{A}'}$ in the updated auxiliary structure $\mathcal{A}'$ consists of all tuples $\bar{a}$ such that $\varphi_\delta^A(\bar{a})$ is satisfied in the structure $(\mathcal{I}', \mathcal{A})$.

We say that a dynamic program Π *maintains* a dynamic query (Q, Δ), if after applying a sequence α of changes over Δ to an initial structure $\mathcal{I}_0$ and applying the corresponding update programs to $(\mathcal{I}_0, \mathcal{A}_0)$, where $\mathcal{A}_0$ is an initial auxiliary structure, a dedicated auxiliary relation is always equal to the result of evaluating Q on the current input structure. Following Patnaik and Immerman [23], we demand that the initial input structure $\mathcal{I}_0$ is *empty*, so, has empty relations. The initial auxiliary structure is over the same domain as $\mathcal{I}_0$ and is defined from $\mathcal{I}_0$ by some first-order definable initialization.

The class DynFO is the class of all dynamic queries that are maintained by a dynamic program with $\mathsf{FO}(\leq,+,\times)$ formulas as update programs[3]. Equivalently, we can think of the update programs as being AC^0 circuits. The class $\mathsf{DynFO}[f(n)]$ allows for $\mathsf{AC}[f(n)]$ circuits as update programs. We often use the equivalence $\mathsf{AC}[f(n)] = \mathsf{IND}[f(n)]$ and think of update programs that apply an $\mathsf{FO}(\leq,+,\times)$ update formula $f(n)$ times. In this paper, we are particularly interested in the class $\mathsf{DynFOLL} = \mathsf{DynFO}[\log\log n]$.

3 The small-structure technique

The small-structure technique has been used for obtaining maintenance results for DynFO for non-constant size changes [7, 24]. The idea is simple: for changes of size m, (1) compute a structure with a domain of size roughly m, depending on the changes and the current auxiliary data, then (2) compute information about this structure (as $m \ll n$, this computation can be more powerful than AC^0), and (3) combine the result with the current auxiliary data to obtain the new auxiliary data.

For DynFO and changes of polylogarithmic size, one can use SAC^1-computations in step (2), as formalized in the next lemma.

▶ **Lemma 1** ([24, Corollary 3]). *Let Q be a k-ary query on σ-structures, for some $k \in \mathbb{N}$. If Q is uniform SAC^1-computable, then there is an $\mathsf{FO}(\leq,+,\times)$ formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size* $\mathrm{polylog}(n)$ *and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$. Here, $\mathcal{S}[C]$ denotes the substructure of $\mathcal{S}$ induced by C.*

For DynFOLL, this generalizes in two directions: (a) for structures of size $\mathrm{polylog}(n)$ one can use NC^2-computations, (b) for structures of size $(\log n)^{\mathcal{O}(\log\log n)}$ one can use SAC^1-computations. This is captured by the following lemma.

▶ **Lemma 2.** *Let Q be a k-ary query on σ-structures, for some $k \in \mathbb{N}$.*

(a) *If Q is uniform NC^2-computable, then there is an FOLL formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size* $\mathrm{polylog}(n)$ *and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$.*

(b) *If Q is uniform SAC^1-computable, then there is an FOLL formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size $(\log n)^{\mathcal{O}(\log\log n)}$ and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$.*

Proof.

(a) Let C have size m, which is polylogarithmically bounded in n. The NC^2-circuit for Q has polynomial size in m and depth $\mathcal{O}((\log m)^2)$, so its size is polylogarithmic in n and the depth is $\mathcal{O}((\log\log n)^2)$. It is well-known[4] that for every NC-circuit of depth $f(n)$ there is an equivalent AC-circuit of depth $\mathcal{O}(\frac{f(n)}{\log\log n})$ and size polynomial in the original circuit, so we can obtain an AC-circuit for answering Q on C with depth $\mathcal{O}(\log\log n)$.

---

[3] Other papers write DynFO for the class that uses FO update formulas without a priori access to the arithmetic relations $\leq,+,\times$ and $\mathsf{DynFO}(\leq,+,\times)$ for the class that uses $\mathsf{FO}(\leq,+,\times)$ update formulas. If changes only affect single tuples, there is no difference for most interesting queries, see [6, Proposition 7]. For changes that affect sets of tuples of non-constant size, all DynFO maintainability results use $\mathsf{FO}(\leq,+,\times)$ update formulas, as FO update formulas without arithmetic are not strong enough to maintain interesting queries. We therefore just write DynFO and omit the suffix $(\leq,+,\times)$ to avoid visual clutter.

[4] Divide the circuit into layers of depth $\log\log n$. Each layer depends only on $\log n$ gates of the previous layer, as each gate has fan-in at most 2, and can be replaced by a constant-depth circuit for the CNF of the layer, which has polynomial size.

(b) The proof of [1, Lemma 8.1] can easily be extended towards the following statement: if a language L is decided by a non-deterministic Turing machine with polynomial time bound m^c and polylogarithmic space bound $(\log m)^d$ then for every positive, non-decreasing and first-order constructible function $t(n)$ there is a uniform AC circuit family for L with depth $\mathcal{O}(t(n))$ and size $2^{\mathcal{O}(m^{\frac{c}{t(n)}}(\log m)^d)}$. For $m = \log n^{e \log\log n}$ and $t(n) = 2ce \log\log n$, the size is exponential in $\sqrt{\log n}\,(\log\log n)^{\mathcal{O}(1)}$, and therefore, as this function grows slower than $\log n$, polynomial in n. The statement follows as all SAC^1 languages can be decided by a non-deterministic Turing machine with polynomial time bound and space bounded by $(\log n)^2$, see [4]. ◀

A straightforward application of the technique to dynamic programs from the literature yields the following DynFOLL-programs. For directed reachability, adapting the DynFO-program for $\mathcal{O}(\frac{\log n}{\log\log n})$ changes from [9] yields (with more proof details in the full version):

▶ **Theorem 3.** *Reachability in directed graphs is in* DynFOLL *under insertions and deletions of* $\text{polylog}(n)$ *edges.*

For undirected reachability and regular languages, replacing Lemma 1 by Lemma 2(b) in the DynFO maintainability proofs for $\text{polylog}(n)$ changes from [7, 24] directly yields:

▶ **Theorem 4.**

(a) *Reachability in undirected graphs is in* DynFOLL *under insertions and deletions of* $(\log n)^{\mathcal{O}(\log\log n)}$ *edges.*

(b) *Membership in regular languages is in* DynFOLL *under symbol changes at* $(\log n)^{\mathcal{O}(\log\log n)}$ *positions.*

The small-structure technique has further applications beyond graph reachability and regular languages. We mention a few here. The proofs are deferred to the full version.

▶ **Theorem 5.**

(a) *A minimum spanning forest for weighted graphs can be maintained*

  (i) *in* DynFO *under changes of* $\text{polylog}(n)$ *edges, and*

  (ii) *in* DynFOLL *under changes of* $(\log n)^{\mathcal{O}(\log\log n)}$ *edges.*

(b) *A maximal matching can be maintained in* DynFOLL *under changes of* $\text{polylog}(n)$ *edges.*

(c) *For graphs with maximum degree bounded by a constant* δ*, a proper* $(\delta+1)$*-colouring can be maintained in* DynFO *under changes of* $\text{polylog}(n)$ *edges.*

4 The hierarchical technique

In this section we describe and use a simple, yet powerful hierarchical technique for handling polylogarithmic changes in DynFOLL. After changing $m \stackrel{\text{def}}{=} (\log n)^c$ many tuples, auxiliary data $\mathcal{R}^1, \ldots, \mathcal{R}^k$ is built in $k \stackrel{\text{def}}{=} d \log\log n$ rounds, for suitable d. The auxiliary data $\mathcal{R}^{\ell-1}$ after round $\ell - 1$ encodes information for certain subsets of the changes of size $2^{\ell-1}$. This information is then combined, via first-order formulas, to information on 2^ℓ changes in round ℓ. The challenge for each concrete dynamic query is to find suitable auxiliary data which is defined depending on a current instance as well as on subsets of changes, and can be combined via first-order formulas to yield auxiliary data for larger subsets of changes.

We apply this approach to maintaining distances in acyclic and undirected graphs, context-free language membership, and tree-isomorphism under polylogarithmic changes. In these applications of the hierarchical technique, information is combined along paths, binary trees, and arbitrary trees, respectively.

4.1 Undirected and acyclic reachability and distances

The articles that introduced the class DynFO showed that reachability for undirected and for acyclic graphs is in DynFO under single-edge changes [10, 23]. For these classes of graphs, also distances, that is, the number of edges in a shortest path between two reachable nodes, can be maintained. For undirected graphs, this was proven in [17], for acyclic graphs it is a straightforward extension of the proof for reachability from [10].

While reachability for undirected graphs is in DynFO under polylogarithmically many edge changes [7], we only know the general $\mathcal{O}(\frac{\log n}{\log \log n})$ bound for acyclic graphs [9]. It is unknown whether distances can be maintained in DynFO under changes of non-constant size, both for undirected and for directed, acyclic graphs.

▶ **Theorem 6.** *Distances can be maintained in* DynFOLL *under insertions and deletions of* $\text{polylog}(n)$ *edges for*

(a) *undirected graphs,*

(b) *acyclic directed graphs.*

To maintain distances, a dynamic programs can use a relation of the form $\text{DIST}(u, v, d)$ with the meaning "the shortest path from u to v has length d". The proof of Theorem 6 is then a direct application of the hierarchical technique on paths. After inserting polylogarithmically many edges, distance information for two path fragments can be iteratively combined to distance information for paths fragments that involve more changed edges. Thus $\text{polylog}\, n$ path fragments (coming from so many connected components before the insertion) can be combined in $\log \log n$ many iterations.

To handle edge deletions, we observe that some distance information is still guaranteed to be valid after the deletion: the shortest path from u to v surely has still length d after the deletion of some edge e if there was no path of length d from u to v that used e. These "safe" distances can be identified using the DIST relation. We show that after deleting polylogarithmically many edges, shortest paths can be constructed from polylogarithmically many "safe" shortest paths of the original graph. We make this formal now.

▶ **Lemma 7.** *Let $G = (V, E)$ be an undirected or acyclic graph, $e \in E$ an edge and $u, v \in V$ nodes such that there is a path from u to v in $G' = (V, E - e)$. For every shortest path $u = w_0, w_1, \ldots, w_{d-1}, w_d = v$ from u to v in G' there is an edge (w_i, w_{i+1}) such that no shortest path from u to w_i and no shortest path from w_{i+1} to v in the original graph G uses e.*

Proof. For undirected graphs, this was proven in [22, Lemma 3.5c]. We give the similar proof for acyclic graphs. If no node w_{i+1} on a shortest path $u = w_0, w_1, \ldots, w_{d-1}, w_d = v$ from u to v in G' exists such that some shortest path from u to w_{i+1} in G uses the edge e, the edge (w_{d-1}, v) satisfies the lemma statement. Otherwise, let w_{i+1} be the first such node on the path. It holds $i + 1 \geq 1$, as the shortest path from u to u trivially does not use e. So, no shortest path from u to w_i in G uses e. There is no shortest path from w_{i+1} to v in G that uses e: otherwise, there would be a path from e to w_{i+1} and a path from w_{i+1} to e in G, contradicting the assumption that G is acyclic. ◀

▶ **Corollary 8.** *Let $G = (V, E)$ be an undirected or acyclic graph and $\Delta E \subseteq E$ with $|\Delta E| = m$. For all nodes u and v such that v is reachable from u in $G' \stackrel{\text{def}}{=} (V, E \setminus \Delta E)$ there is a shortest path in G' from u to v that is composed of at most m edges and $m + 1$ shortest paths of G, each from some node u_i to some node v_i for $i \leq m + 1$, such that no shortest path from u_i to v_i in G uses an edge from ΔE.*

Proof idea. Via induction over m. For $m = 1$, this follows from Lemma 7. For $m > 1$ this is immediate from the induction hypothesis. ◀

We can now prove that distances in undirected and in acyclic graphs can be maintained in DynFOLL under changes of polylogarithmic size.

Proof of Theorem 6. We construct a DynFOLL program that maintains the auxiliary relation $\textsc{dist}(u, v, d)$ with the meaning "the shortest path from u to v has length d".

Suppose $m \stackrel{\text{def}}{=} (\log n)^c$ edges ΔE are changed in $G = (V, E)$ yielding the graph $G' = (V, E')$. W.l.o.g. all edges in ΔE are either inserted or deleted. In both cases, the program executes a first-order initialization, yielding auxiliary relations $\textsc{dist}^0(u, v, d)$, and afterwards executes a first-order procedure for $k \stackrel{\text{def}}{=} c \log\log n$ rounds, yielding auxiliary relations $\textsc{dist}^1, \dots, \textsc{dist}^k$. The superscripts on the relations are for convenience, they are all subsequently stored in $\textsc{dist}$.

For insertions, we use the standard inductive definition of reachability and distances. Set $\textsc{dist}^0(u, v, d) \stackrel{\text{def}}{=} \textsc{dist}(u, v, d)$, where $\textsc{dist}(u, v, d)$ is the distance information of the unchanged graph G. Then, for k rounds, the distance information is combined with the new edges ΔE, doubling the amount of used edges from ΔE in each round. Thus $\textsc{dist}^\ell(u, v, d)$ is computed from $\textsc{dist}^{\ell-1}$ by including $\textsc{dist}^{\ell-1}$ and all tuples which satisfy the formula:

$$\varphi_{\textsc{ins}} \stackrel{\text{def}}{=} \exists z_1 \exists z_2 \exists d_1 \exists d_2 (\Delta E(z_1, z_2) \wedge d_1 + d_2 + 1 = d \wedge \textsc{dist}^{\ell-1}(u, z_1, d_1) \wedge \textsc{dist}^{\ell-1}(z_2, v, d_2))$$

For deletions, the program starts from shortest paths $u, \dots, v$ in G such that no shortest path from u to v uses edges from ΔE and then combines them for k rounds, which yields the correct distance information for G' according to Corollary 8. Thus, the first-order initialization yields $\textsc{dist}^0(u, v, d)$ via

$$\textsc{dist}(u, v, d) \wedge \neg \exists z_1 \exists z_2 \exists d_1 \exists d_2 (d = d_1 + d_2 + 1 \wedge \Delta E(z_1, z_2) \wedge \textsc{dist}(u, z_1, d_1) \wedge \textsc{dist}(z_2, v, d_2))$$

Then $\textsc{dist}^\ell(u, v, d)$ is computed from $\textsc{dist}^{\ell-1}(u, v, d)$ via a formula similar to $\varphi_{\textsc{ins}}$, using E instead of ΔE. ◀

4.2 Context-free language membership

Membership problems for formal languages have been studied in dynamic complexity starting with the work of Gelade, Marquardt, and Schwentick [15]. It is known that context-free languages can be maintained in DynFO under single symbol changes [15] and that regular languages can even be maintained under polylog changes [25, 24].

It is an open problem whether membership in a context-free language can be maintained in DynFO for changes of non-constant size. We show that this problem is in DynFOLL under changes of polylogarithmic size.

▶ **Theorem 9.** *Every context-free language can be maintained in* DynFOLL *under changes of size* polylog n.

Suppose $G = (V, \Sigma, S, \Gamma)$ is a grammar in Chomsky normal form with $L \stackrel{\text{def}}{=} L(G)$. For single changes, 4-ary auxiliary relations $R_{X \to Y}$ are used for all $X, Y \in V$ [15], with the intention that $(i_1, j_1, j_2, i_2) \in R_{X \to Y}$ iff $X \Rightarrow^* w[i_1, j_1) Y w(j_2, i_2]$, where $w \stackrel{\text{def}}{=} w_1 \dots w_n$ is the current string. Let us call $I = (i_1, j_1, j_2, i_2)$ a *gapped interval*. For a gapped interval I and a set P of changed positions, denote by $\#(I, P)$ the number of changed positions $p \in P$ with $p \in [i_1, j_1) \cup (j_2, i_2]$.

The idea for the DynFOLL program for handling polylog changes is simple and builds on top of the program for single changes. It uses the same auxiliary relations and, after changing a set P of positions, it collects gapped intervals I into the relations $R_{X\to Y}$ for increasing $\#(I,P)$ in at most $\mathcal{O}(\log\log n)$ rounds. Initially, gapped intervals with $\#(I,P)\leq 1$ are collected using the first-order update formulas for single changes. Afterwards, in each round, gapped intervals I with larger $\#(I,P)$ are identified by splitting I into two gapped intervals I_1 and I_2 with $\#(I,P)=\#(I_1,P)+\#(I_2,P)$ such that I can be constructed from I_1 and I_2 with a first-order formula.

To ensure that $\mathcal{O}(\log\log n)$ many rounds suffice, we need that the intervals I_1 and I_2 can always be chosen such that $\#(I_1,P)$ and $\#(I_2,P)$ are of similar size. This will be achieved via the following simple lemma, which will be applied to parse trees. For a binary tree $T=(V,E)$ with red coloured nodes $R\subseteq V$, denote by $\#(T,R)$ the number of red nodes of T. For a tree T and a node v, let T_v be the subtree of T rooted at v.

▶ **Lemma 10.** *For all rooted binary trees $T=(V,E,r)$ with red coloured nodes $R\subseteq V$, there is a node $v\in V$ such that:*
- *$\#(T_v,R)\leq\frac{2}{3}\cdot\#(T,R)$ and*
- *$\#(T\setminus T_v,R)\leq\frac{2}{3}\cdot\#(T,R)$*

Proof idea. Walk down the tree starting from its root by always choosing the child whose subtree contains more red coloured nodes. Stop as soon as the conditions are satisfied. ◀

We now provide the detailed proof of Theorem 9.

Proof (of Theorem 9). We construct a DynFOLL program that maintains the auxiliary relations $R_{X\to Y}$ for all $X,Y\in V$. Suppose $m\overset{\text{def}}{=}(\log n)^c$ positions P are changed. The program executes a first-order initialization, yielding auxiliary relations $R^0_{X\to Y}$, and afterwards executes a first-order procedure for $k\overset{\text{def}}{=}d\log\log n$ rounds, for $d\in\mathbb{N}$ chosen such that $(\frac{3}{2})^k>m$, yielding auxiliary relations $R^1_{X\to Y},\dots,R^k_{X\to Y}$. The superscripts on the relations are for convenience, they are all subsequently stored in $R_{X\to Y}$.

For initialization, the DynFOLL program includes gapped intervals (i_1,j_1,j_2,i_2) into the relations $R^0_{X\to Y}$ for which
- no position in $[i_1,j_1)\cup(j_2,i_2]$ has changed and (i_1,j_1,j_2,i_2) was previously in $R_{X\to Y}$, or
- exactly one position in $[i_1,j_1)\cup(j_2,i_2]$ has changed and the dynamic program for single changes from [15] includes the tuple (i_1,j_1,j_2,i_2) into $R_{X\to Y}$.

Afterwards, for k rounds, the DynFOLL program applies the following first-order definable procedure to its auxiliary relations. A gapped interval $I=(i_1,j_1,j_2,i_2)$ is included into $R^\ell_{X\to Y}$ in round ℓ if it was included in $R^{\ell-1}_{X\to Y}$ or one of the following conditions hold (see Figure 1 for an illustration):

(a) There are gapped intervals $I_1=(i_1,u_1,u_2,j_2)$ and $I_2=(u_1,j_1,j_2,u_2)$ and a non-terminal $Z\in V$ such that $I_1\in R^{\ell-1}_{X\to Z}$ and $I_2\in R^{\ell-1}_{Z\to Y}$. This can be phrased as first-order formula as follows:

$$\varphi_a\overset{\text{def}}{=}\exists u_1,u_2\Big[(i_1\leq u_1\leq j_1\leq j_2\leq u_2\leq i_2)\wedge \bigvee_{Z\in V}\Big(R_{X\to Z}(i_1,u_1,u_2,i_2)\wedge R_{Z\to Y}(u_1,j_1,j_2,u_2)\Big)\Big]$$

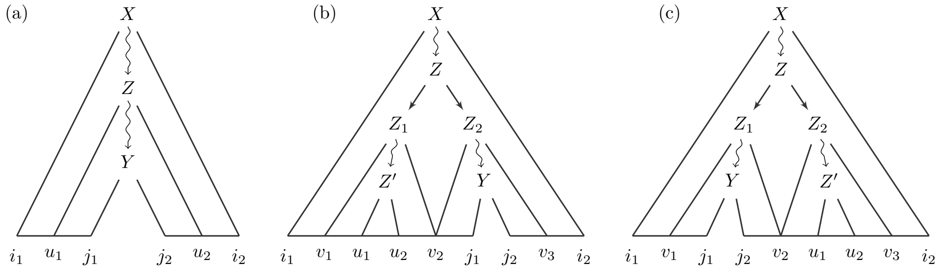

Figure 1 Illustration of when a gapped interval (i_1, j_1, j_2, i_2) is added to $R_{X \to Y}$ in the proof of Theorem 9.

(b) There are gapped intervals $I_1 = (i_1, v_1, v_3, i_2)$, $I_2 = (v_1, u_1, u_2, v_2)$, and $I_3 = (v_2, j_1, j_2, v_3)$ and non-terminals $Z, Z_1, Z_2, Z' \in V$ such that $Z \to Z_1 Z_2 \in \Gamma$ and $I_1 \in R^{\ell-1}_{X \to Z}$, $I_2 \in R^{\ell-1}_{Z_1 \to Z'}$, $I_3 \in R^{\ell-1}_{Z_2 \to Y}$ and $w[u_1, u_2]$ can be derived from Z'. This can be phrased as first-order formula as follows:

$$\varphi_b \stackrel{\text{def}}{=} \exists u_1, u_2, v_1, v_2, v_3 \Bigg[(i_1 \le v_1 \le u_1 \le u_2 \le v_2 \le j_1 \le j_2 \le v_3 \le i_2) \wedge \bigvee_{\substack{Z, Z_1, Z_2, Z' \in V \\ Z \to Z_1 Z_2 \in \Gamma}} \Big(R_{X \to Z}(i_1, v_1, v_3, i_2) \wedge R_{Z_1 \to Z'}(v_1, u_1, u_2, v_2) \wedge R_{Z'}(u_1, u_2) \wedge R_{Z_2 \to Y}(v_2, j_1, j_2, v_3) \Big) \Bigg]$$

Here, $R_{Z'}(u_1, u_2)$ is an abbreviation for the formula stating that $w[u_1, u_2]$ can be derived from Z', i.e. $R_{Z'}(u_1, u_2) \stackrel{\text{def}}{=} \exists v \bigvee_{W \to \sigma \in \Gamma} (R_{Z' \to W}(u_1, v, v, u_2) \wedge \sigma(v))$.

(c) Symmetrical to (b), with gapped intervals $I_1 = (i_1, v_1, v_3, i_2)$, $I_2 = (v_1, j_1, j_2, v_2)$, and $I_3 = (v_2, u_1, u_2, v_3)$ and non-terminals $Z, Z_1, Z_2, Z' \in V$ such that $Z \to Z_1 Z_2 \in \Gamma$ and $I_1 \in R^{\ell-1}_{X \to Z}$, $I_2 \in R^{\ell-1}_{Z_1 \to Y}$, $I_3 \in R^{\ell-1}_{Z_2 \to Z'}$ and $w[u_1, u_2]$ can be derived from Z'.

Note that $\#(I, P) = \#(I_1, P) + \#(I_2, P)$ in case (a) and $\#(I, P) = \#(I_1, P) + \#(I_2, P) + \#(I_3, P)$ in cases (b) and (c). Using Lemma 10, the intervals I_j can be chosen such that $\#(I_j, P) \le \frac{2}{3} \cdot \#(I, P)$ and thus k rounds suffice. ◀

It is known that for single tuple changes one can maintain for edge-labeled, acyclic graphs whether there is a path between two nodes with a label sequence from a fixed context-free language [21]. The techniques we have seen can be used to also lift this result to changes of polylogarithmic size.

▶ **Proposition 11.** *Context-free path queries can be maintained under changes of polylogarithmic size in* DynFOLL *on acyclic graphs.*

4.3 Tree isomorphism

The dynamic tree isomorphism problem – given a forest $F = (V, E)$ and two nodes $x, x^* \in V$, are the subtrees rooted at x and x^* isomorphic? – has been shown to be maintainable in DynFO under single edge insertions and deletions by Etessami [14].

It is not known whether tree isomorphism can be maintained in DynFO under changes of size $\omega(1)$. We show that it can be maintained in DynFOLL under changes of size polylog n:

▶ **Theorem 12.** *Tree isomorphism can be maintained in* DynFOLL *under insertion and deletion of* $\mathrm{polylog}\, n$ *edges.*

Intuitively, we want to use Etessami's dynamic program as the base case for a DynFOLL-program: (1) compute isomorphism information for pairs of subtrees in which only one change happened, then (2) combine this information in $\log \log n$ many rounds. Denote by $\mathrm{subtree}_x(r)$ the subtree rooted at r, in the tree rooted at x, within the forest F. The main ingredient in Etessami's program is a 4-ary auxiliary relation T-ISO for storing tuples (x, r, x^*, r^*) such that $\mathrm{subtree}_x(r)$ and $\mathrm{subtree}_{x^*}(r^*)$ are isomorphic and disjoint in F. It turns out that this information is not "composable" enough for step (2).

We therefore slightly extend the maintained auxiliary information. A *(rooted) context* $C = (V, E, r, h)$ is a tree (V, E) with root $r \in V$ and one distinguished leaf $h \in V$, called the *hole*. Two contexts $C = (V, E, r, h)$, $C^* = (V^*, E^*, r^*, h^*)$ are isomorphic if there is a root- and hole-preserving isomorphism between them, i.e. an isomorphism that maps r to r^* and h to h^*. For a forest F and nodes x, r, h occurring in this order on some path, the context $C(x, r, h)$ is defined as the context we obtain by taking $\mathrm{subtree}_x(r)$, removing all children of h, and taking r as root and h as hole. Our dynamic program uses

- a 6-ary auxiliary relation C-ISO for storing tuples $(C, C^*) \stackrel{\text{def}}{=} (x, r, h, x^*, r^*, h^*)$ such that the contexts $C \stackrel{\text{def}}{=} C(x, r, h)$ and by $C^* \stackrel{\text{def}}{=} C(x^*, r^*, h^*)$ are disjoint and isomorphic,
- a ternary auxiliary relation DIST for storing tuples (x, y, d) such that the distance between nodes x and y is d, and
- a 4-ary auxiliary relation #ISO-SIBLINGS for storing tuples (x, r, y, m) such that y has m isomorphic siblings within $\mathrm{subtree}_x(r)$.

The latter two relations have also been used by Etessami. From distances, a relation PATH(x, y, z) with the meaning "y is on the unique path between x and z" is FO-definable on forests, see [14]. The relation T-ISO(x, r, x^*, r^*) can be FO-defined from C-ISO.

We will now implement the steps (1) and (2) with these adapted auxiliary relations. Suppose a forest $F \stackrel{\text{def}}{=} (V, E)$ is changed into the forest $F' \stackrel{\text{def}}{=} (V, E')$ by changing a set ΔE of edges. A node $v \in V$ is *affected* by the change, if v is adjacent to some edge in ΔE. The DynFOLL program iteratively collects isomorphic contexts C and C^* of F' with more and more affected nodes. Denote by $\#(C, C^*, \Delta E)$ the number of nodes in contexts C and C^*, excluding hole nodes, affected by change ΔE.

The following lemma states that C-ISO can be updated for pairs of contexts with at most one affected node each. Its proof is very similar to Etessami's proof.

▶ **Lemma 13.** *Given* C-ISO*,* DIST*, and* #ISO-SIBLINGS *and a set of changes* ΔE*, the set of pairs* (C, C^*) *of contexts such that* C, C^* *are disjoint and isomorphic and such that both* C *and* C^* *contain at most one node affected by* ΔE *is* FO*-definable.*

The dynamic program will update the auxiliary relation C-ISO for contexts with at most one affected node per context using Lemma 13. Isomorphic pairs (C, C^*) of contexts with larger $\#(C, C^*, \Delta E)$ are identified by splitting both C and C^* into smaller contexts.

The splitting is done such that the smaller contexts have fewer than $\frac{2}{3} \cdot \#(C, C^*, \Delta E)$ affected nodes. To this end, we will use the following simple variation of Lemma 10. For a tree $T = (V, E)$ and a function $p : V \to \{0, 1, 2\}$ which assigns each a node number of pebbles, let $\#(T, p)$ be the total number of pebbles assigned to nodes in T.

▶ **Lemma 14.** *Let* T *be a tree of unbounded degree and* p *such that either (i)* $\#(T, p) > 2$*, or (ii)* $\#(T, p) = 2$ *and* $p(v) \le 1$ *for all* $v \in V$*. Then there is a node* v *such that either:*

(1) $\#(T \setminus T_v, p) \le \frac{2}{3} \cdot \#(T, p)$ *and* $\#(T_v, p) \le \frac{2}{3} \cdot \#(T, p)$*, or*

(2) $\#(T \setminus T_v, p) \le \frac{1}{3} \cdot \#(T, p)$ *and* $\#(T_u, p) \le \frac{1}{3} \cdot \#(T, p)$ *for any child* u *of* v*.*

Proof idea. Use the same approach as for Lemma 10. If no node v of type (1) is found, a node of type (2) must exist. ◀

We now prove that tree isomorphism can be maintained in DynFOLL under changes of polylogarithmic size.

Proof (of Theorem 12). We construct a DynFOLL program that maintains the auxiliary relations C-ISO, DIST, and #ISO-SIBLINGS. Suppose $m \stackrel{\text{def}}{=} (\log n)^c$ edges ΔE are changed. As a preprocessing step, the auxiliary relation DIST is updated in depth $\mathcal{O}(\log\log n)$ via Theorem 6. Then, C-ISO is updated by first executing a first-order initialization for computing an initial version C-ISO^0. Afterwards a first-order procedure is executed for $k \stackrel{\text{def}}{=} d\log\log n$ rounds, for $d \in \mathbb{N}$ chosen such that $(\frac{3}{2})^k > m$, yielding auxiliary relations $\{\text{C-ISO}^\ell\}_{\ell\leq k}$ and $\{\#\text{ISO-SIBLINGS}^\ell\}_{\ell\leq k}$. The superscripts on the relations are for convenience, they are all subsequently stored in C-ISO, DIST, and #ISO-SIBLINGS.

The goal is that after the ℓth round

- C-ISO$^\ell$ contains all pairs C, C^* with $\#(C, C^*, \Delta E) \leq (\frac{3}{2})^\ell$ which are isomorphic and disjoint, and
- #ISO-SIBLINGS$^\ell$ contains the number of isomorphic siblings identified so far (i.e., with respect to C-ISO$^\ell$).

Round ℓ first computes C-ISO$^\ell$ with a first-order procedure, and afterwards computes #ISO-SIBLINGS$^\ell$. For initialization, the DynFOLL program first computes C-ISO^0, using Lemma 13, and #ISO-SIBLINGS^0. Afterwards, for k rounds, the DynFOLL program combines known pairs of isomorphic contexts into pairs with more affected nodes and adapts C-ISO and #ISO-SIBLINGS accordingly.

Computing C-ISO$^\ell$. In the ℓth round, the program tests whether contexts $C \stackrel{\text{def}}{=} C(x, r, h)$ and $C^* \stackrel{\text{def}}{=} C(x^*, r^*, h^*)$ with $\#(C, C^*, \Delta E) \leq (\frac{3}{2})^\ell$ are isomorphic by splitting both C and C^* into contexts with fewer affected nodes. The splitting is done by selecting suitable nodes $z \in C$ and $z^* \in C^*$, and splitting the context depending on these nodes.

We first provide some intuition of how z and z^* are intended to be chosen. Suppose C and C^* are isomorphic via isomorphism π. With the goal of applying Lemma 14, let p be the function that assigns to each non-hole node v of C a number of pebbles from $\{0, 1, 2\}$ indicating how many of the two nodes v and $\pi(v)$ have been affected by the change ΔE. Note that if (C, p) does not fulfill the precondition of Lemma 14, then (C, C^*) must have already been included in C-ISO^0 during the initialization. Therefore, assume the precondition holds for (C, p). Let C_z denote the subcontext of C rooted at z. Now, applying Lemma 14 to (C, p) yields a node z such that one of the following cases holds:

(1) $\#(C \setminus C_z, p) \leq \frac{2}{3} \cdot \#(C, p)$ and $\#(C_z, p) \leq \frac{2}{3} \cdot \#(C, p)$, or
(2) $\#(C \setminus C_z, p) \leq \frac{1}{3} \cdot \#(C, p)$ and $\#(C_u, p) \leq \frac{1}{3} \cdot \#(C, p)$ for any child u of z.

Intuitively our first-order procedure tries to guess this node z and its image $z^* \stackrel{\text{def}}{=} \pi(z)$ and split the contexts C and C^* at these nodes.

For testing that C and C^* are isomorphic, the program guesses two nodes z and z^* and (disjunctively) chooses case (1) or (2). Note that the program cannot be sure that it has correctly guessed z and z^* according to the above intuition. For this reason, the program first tests that the size restrictions from the chosen case (1) or (2) are fulfilled, which is easily possible in FO$(\leq, +, \times)$ as there are at most polylogarithmically many affected nodes. Since $\#(C, C^*, \Delta E) \leq \left(\frac{3}{2}\right)^\ell$, this ensures that, by induction, C-ISO$^{\ell-1}$ is fully correct for all pairs of contexts that will be compared when testing isomorphism of C and C^*. Note that in case (2) the subtrees of any pair of children u_1 and u_2 of z have at most $\frac{2}{3} \cdot \#(C, C^*, \Delta E)$ affected nodes.

Next, the procedure tests that there is an isomorphism between C and C^* that maps z to z^*. The following claim is used:

▷ Claim 15. Suppose z and z^* are nodes in C and C^* that satisfy Condition (1) or (2) with children $Z \uplus Y$ and $Z^* \uplus Y^*$, respectively, with $|Y| = |Y^*|$ constant. Then a first-order formula can test whether there is an isomorphism between the forests $\{\text{subtree}_z(u) \mid u \in Z\}$ and $\{\text{subtree}_{z^*}(u^*) \mid u^* \in Z^*\}$ using the relations $\textsc{c-iso}^{\ell-1}$ and $\#\textsc{iso-siblings}^{\ell-1}$.

Proof. The forests are isomorphic iff for each $u \in Z$ there is a $u^* \in Z^*$ such that $\text{subtree}_z(u) \cong \text{subtree}_{z^*}(u^*)$ and such that the number of nodes $v \in Z$ with $\text{subtree}_z(u) \cong \text{subtree}_z(v)$ is the same as the number of nodes v^* with subtrees $\text{subtree}_{z^*}(u^*) \cong \text{subtree}_{z^*}(v^*)$, and vice versa with roles of u and u^* swapped.

Because z and z^* satisfy condition (1) or (2), $\textsc{c-iso}^{\ell-1}$ is correct on the forest $\{\text{subtree}_z(u) \mid u \in Z\} \cup \{\text{subtree}_{z^*}(u^*) \mid u^* \in Z^*\}$ by induction. Additionally, $\#\textsc{iso-siblings}^{\ell-1}$ is consistent with $\textsc{c-iso}^{\ell-1}$ by induction. From $\textsc{c-iso}^{\ell-1}$, the tree isomorphism relation $\textsc{t-iso}^{\ell-1}$ – storing tuples (x, r, x^*, r^*) such that $\text{subtree}_x(r)$ and $\text{subtree}_{x^*}(r^*)$ are isomorphic and disjoint – is FO-definable.

For testing whether a node $u \in Z$ satisfies the above condition, a first-order formula existentially quantifies a node $u^* \in Z^*$ and checks that $\textsc{t-iso}^{\ell-1}(z, u, z^*, u^*)$. The number of isomorphic siblings of u, u^* is compared using $\#\textsc{iso-siblings}^{\ell-1}$ and subtracting any siblings $y \in Y$ for which $\textsc{t-iso}^{\ell-1}(z, u, z, y)$ (and, respectively, $y^* \in Y^*$ for which $\textsc{t-iso}^{\ell-1}(z^*, u^*, z^*, y^*)$). This is possible in FO because (a) $|Y|$ is constant and (b) $\#\textsc{iso-siblings}^{\ell-1}$ is consistent with $\textsc{c-iso}^{\ell-1}$ (even though $\textsc{c-iso}^{\ell-1}$ is not necessarily complete on subtrees in Y, Y^*). ◁

For testing whether there is an isomorphism between C and C^* mapping z to z^*, the program distinguishes the cases (1) and (2) from above. Further, in each of the cases it distinguishes (A) $\textsc{path}(r, z, h)$ and $\textsc{path}(r^*, z^*, h^*)$, or (B) $\neg\textsc{path}(r, z, h)$ and $\neg\textsc{path}(r^*, z^*, h^*)$. For all these first-order definable cases, a first-order formula can test whether there is an isomorphism between C and C^* mapping z to z^*. The detailed case analysis is deferred to the full version of the paper.

Computing $\#$ISO-SIBLINGS$^\ell$. The relation $\#\textsc{iso-siblings}^\ell$ can be first-order defined from $\textsc{c-iso}^\ell$ and the 4-ary relation $\#\textsc{iso-sibling}_{\text{unchanged}}$ containing tuples

- (x, r, y, m) with $m > 0$ if $\text{subtree}_x(y)$ has no affected nodes and the number of isomorphic siblings of y in $\text{subtree}_x(r)$ with no affected nodes is m; and
- $(x, r, y, 0)$ if $\text{subtree}_x(y)$ contains an affected node.

Thus the relation $\#\textsc{iso-sibling}_{\text{unchanged}}$ contains isomorphism counts for "unchanged" siblings. It is FO-definable from the old auxiliary data (from before the change) and the set of changes.

Given $\#\textsc{iso-sibling}_{\text{unchanged}}$ and $\textsc{c-iso}^\ell$, the relation $\#\textsc{iso-sibling}^\ell$ is FO-definable as follows. Include a tuple (x, r, y, m) into $\#\textsc{iso-sibling}^\ell$ if $m = m_1 + m_2$ where m_1 is the number of unchanged, isomorphic siblings of y and m_2 is the number of isomorphic siblings of y affected by the change (but by at most $(\frac{3}{2})^{\ell-1}$ changes).

The number m_1 can be checked via distinguishing whether $\text{subtree}_x(y)$ has changed or not. If $\text{subtree}_x(y)$ has not changed, the formula checks that m_1 is such that $(x, r, y, m_1) \in \#\textsc{iso-sibling}_{\text{unchanged}}$. If $\text{subtree}_x(y)$ has changed, find a sibling y^* of y with an unchanged, isomorphic subtree. If y^* exists then the formula checks that m_1 is such that $(x, r, y^*, m_1) \in \#\textsc{iso-sibling}_{\text{unchanged}}$, and otherwise that m_1 is 0.

For checking m_2, let $S(y)$ be the set of siblings y^* of y in subtree$_x(r)$ that contain at least one affected node and where T-ISO$^\ell(x, y, x, y^*)$. Since there are at most polylog n changes, $|S(y)| = \mathcal{O}(\text{polylog}\, n)$. Therefore, $|S(y)|$ can be counted and compared to m_2 in FO. ◀

5 Tree decompositions of bounded-treewidth graphs

One of the best-known algorithmic meta-theorems is Courcelle's theorem, which states that all graph properties expressible in monadic second-order logic MSO can be decided in linear time for graphs with tree-width bounded by some constant k [5]. The tree-width is a graph parameter and measures how "tree-like" a graph is and is defined via tree decompositions, see below for details. Courcelle's theorem is based on Bodlaender's theorem, stating that in linear time (1) one can decide whether a graph has tree-width at most k and (2) one can compute a corresponding tree decomposition [3].

Elberfeld, Jakoby and Tantau [13] proved variants of these results and showed that "linear time" can be replaced with "logarithmic space" in both theorem statements. A dynamic version of Courcelle's theorem was proven in [8]: every MSO-definable graph property is in DynFO under changes of single edges. The proof of the latter result circumvented providing a dynamic variant of Bodlaender's theorem, by using the result of Elberfeld et al. that tree decompositions can be computed in LOGSPACE, showing that a tree decomposition can be used to decide the graph property if only logarithmically many single-edge changes occurred after its construction, and that this is enough for maintenance in DynFO.

It is an open problem to generalize the DynFO maintenance result of [8] from single-edge changes to changes of polylogarithmically many edges, even for DynFOLL. Here, we provide an intermediate step and show that tree decompositions for graphs of bounded treewidth can be maintained in DynFOLL. This result may lead to a second strategy for maintaining MSO properties dynamically, in addition to the approach of [8].

A *tree decomposition* (T, B) of a graph $G = (V, E)$ consists of a rooted tree T and a mapping B from the nodes of T to subsets of V. For a tree node t, we call the set $B(t)$ the *bag* of t. A tree decomposition needs to satisfy three conditions. First, every vertex $v \in V$ needs to be included in some bag. Second, for every edge $(u, v) \in E$ there needs to be bag that includes both u and v. Third, for each vertex $v \in V$, the nodes t of T such that $v \in B(t)$ form a connected subgraph in T. The *width* of a tree decomposition is the maximal size of a bag $B(t)$, over all tree nodes t, minus 1. The *treewidth* of a graph G is the minimal width of a tree decomposition for G.

In addition to the width, important parameters of a tree decomposition are its *depth*, the maximal distance from the root to a leaf, and its *degree*, the degree of the tree T. Often, a binary tree decomposition of depth $\mathcal{O}(\log |V|)$ is desirable, while width $\mathcal{O}(k)$ for a graph of treewidth k is tolerable. We show that one can maintain in DynFOLL a tree decomposition of logarithmic depth but with unbounded degree. The proof is deferred to the full version of the paper. It does not use the hierarchical technique; a tree decomposition is defined in FOLL from auxiliary information that is maintained in DynFO.

▶ **Theorem 16.** *For every k, there are numbers $c, d \in \mathbb{N}$ such that a tree decomposition of width ck and depth $d \log n$ can be maintained in* DynFOLL *under changes of* polylog(n) *edges for graphs of treewidth k, where n is the size of the graph.*

6 Conclusion and discussion

We have shown that most existing maintenance results for DynFO under single tuple changes can be lifted to DynFOLL for changes of polylogarithmic size. A notable exception are queries expressible in monadic second-order logic, which can be maintained on graphs of bounded treewidth under single-tuple changes.

Thus it seems very likely that one can find large classes of queries such that: If a query from the class can be maintained in DynFO for changes of size $\mathcal{O}(1)$, then it can be maintained in DynFOLL for polylogarithmic changes. Identifying natural such classes of queries is an interesting question for future research.

References

1 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

2 David A. Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. *J. Comput. Syst. Sci.*, 63(2):186–200, 2001. doi:10.1006/JCSS.2001.1764.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

4 Stephen A. Cook. Path systems and language recognition. In Patrick C. Fischer, Robert Fabian, Jeffrey D. Ullman, and Richard M. Karp, editors, *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing, May 4-6, 1970, Northampton, Massachusetts, USA*, pages 70–72. ACM, 1970. doi:10.1145/800161.805151.

5 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.

6 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.

7 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.122.

8 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:12)2019.

9 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 120:1–120:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.120.

10 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages*, pages 295–308, 1993.

11 Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Inf. Comput.*, 120(1):101–106, 1995. doi:10.1006/inco.1995.1102.

12 Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998. doi:10.1006/jcss.1998.1565.

13 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

14 Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS*, pages 235–243, 1998. doi:10.1145/275487.275514,db/conf/pods/Etessami98.html.

15 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19:1–19:36, 2012. doi:10.1145/2287718.2287719.

16 Andrew V. Goldberg and Serge A. Plotkin. Parallel $(\Delta + 1)$-coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, 1987. doi:10.1016/0020-0190(87)90169-4.

17 Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012. doi:10.1145/2274576.2274601.

18 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

19 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

20 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-24508-4.

21 Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. Dynamic graph queries. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICDT.2016.14.

22 Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698–721, 2005. doi:10.1145/1093382.1093384.

23 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

24 Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of regular languages: Big changes, small work. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CSL.2023.35.

25 Nils Vortmeier. *Dynamic expressibility under complex changes*. PhD thesis, TU Dortmund University, Germany, 2019. doi:10.17877/DE290R-20434.

26 Nils Vortmeier and Thomas Zeume. Dynamic complexity of parity exists queries. *Log. Methods Comput. Sci.*, 17(4), 2021. doi:10.46298/LMCS-17(4:9)2021.