Dynamic Tree Isomorphism via First-order Updates to a Relational Database

Kousha Etessami*
Bell Labs
600 Mountain Ave.
Murray Hill, NJ 07974
email: kousha@research.bell-labs.com

Abstract

In databases, as in other computational settings, one would like to efficiently update answers to queries while minor changes are being made to the data. Dynamic complexity asks what resources are required to perform such updates.

In this paper our main focus will be a particular dynamic graph problem, tree isomorphism, and the efficiency of our update scheme will be measured in terms of how expressive a query language is required to express our update. Working in the framework developed by [DS93, PI97] for dynamic query evaluation, we show that dynamic tree isomorphism can be performed via first-order updates to a relational database (in [DS93] this framework is called a *first-order incremental evaluation system*, and in [PI97] it is called *Dyn-FO*).

In [EI95] it was shown that tree-isomorphism can not be expressed in first-order logic augmented with a transitive closure operator and counting, ($FO + TC +$ COUNT) (but without ordering). We thus obtain a first example of a graph problem in Dyn-FO that is not expressible in this logic. Part of our proof shows how to build and maintain arithmetic predicates on a relevant part of the universe, from which we obtain a direct correspondence between Dyn-FO and dynamic constant parallel time. This provides some explanation for why it is so difficult to prove lower bounds for Dyn-FO.

1 Introduction

Dynamic complexity considers the cost of computation when, instead of performing computation on some fixed input once, minor modifications are repeatedly being made to the input and one needs to update the result of the computation after each change. We can keep track of any kind of information during the history of modifications, and we can use this auxiliary information to perform updates more efficiently.

A great deal of modern computation, most prominently database queries, but also program compilation, web page access, and much more, is performed in a dynamic setting where minor changes are being made to the input. The practical benefits to not recomputing everything from scratch after each change are obvious. In order to be able to rigorously analyze dynamic complexity we need suitable mathematical models for dynamic computation. It is not yet clear which model(s) will be judged appropriate, but several have been proposed. (See, e.g., [MSVT94]. See [PI97] for a brief summary of proposed models and references.)

In the context of databases, there is by now substantial literature on dynamic query evaluation, a.k.a., *incremental view maintenance* (see, e.g., [BLT86, QW91, DS93, GMS93, PI97, DS95, GL95, GM95, Via97, DS97a]). Much of this work (e.g., [GMS93]) is concerned with efficiently recomputing views when the query itself is expressible in a restricted language, such as SQL. In this paper our concern is somewhat different. The goal is to evaluate various queries which aren't necessarily expressible statically in a desired language, but we would like to know if nevertheless the queries can be *updated* dynamically using the desired language. A framework to address precisely this was proposed by [DS93, PI97]. To model dynamic query evaluation both [DS93] and [PI97] proposed using first-order logic as a dynamic query/update language. Their model is structured as follows:

- Input and auxiliary information are stored in relations of fixed arity over a universe assumed to be large enough to incorporate all future updates.
- Modifications to the input occur in the form of insertion or deletion (of a tuple) to/from an input relation.
- Updates are performed using (first-order) formulas over input and auxiliary relations.[1]

*Phone: (908) 582-6431; Fax: (908)582-5857. This research was largely conducted while the author was at BRICS, University of Aarhus, Denmark.

[1] [PI97] provide a framework in which the power of the update com-

- Answers to queries are also obtained via first-order formulas over the input and auxiliary relations.

In [DS93] this model was titled a *First-order Incremental Evaluation System* (**FOIES**) and in [PI97] it was called *Dynamic FO* (**Dyn-FO**). The two proposed models are essentially the same, and we opt for the name Dyn-FO.

The advantages of their model include both its elegance and its direct connection to relational database query languages and relational calculus. It is appealing to be able to dynamically update queries using a simple and well-understood query language such as first-order logic (relational calculus). In general, such a query language is not expressive enough to evaluate many important queries statically from scratch. This would appear to necessitate the use of a more powerful query language or even external programming constructs to evaluate such queries. However, the simple query language often *can update* the query dynamically, and this is frequently all that is needed in the database setting.

Indeed, [PI97, DS95] showed that various queries which are known not to be expressible in first-order logic, are nevertheless in Dyn-FO. Dong and Su [DS95] showed that directed acyclic graph st-connectivity can be performed in Dyn-FO, while Patnaik and Immerman [PI97] showed that undirected graph connectivity and several related graph problems can be performed in Dyn-FO. It remains open whether general directed st-connectivity and transitive closure can be performed in Dyn-FO. Moreover, Dyn-FO is trivially contained in P-time, but it is not yet even known whether this containment is strict.

We show that tree isomorphism can be performed in Dyn-FO. In the dynamic tree isomorphism problem, a forest is being modified repeatedly with the addition and deletion of edges, with the guarantee that no cycles will be introduced. The query to be maintained is whether the trees rooted at a given pair of vertices are isomorphic. It was shown in [EI95] that tree isomorphism is not expressible using first-order logic augmented with a transitive closure operator (FO + TC) without ordering, not even when counting quantifiers are present. Thus, a consequence of our upper bound is that Dyn-FO $\not\subseteq$ (FO + TC + COUNT). With ordering, (FO + TC) corresponds to NL, non-deterministic logarithmic space. Lindell [Lin92] showed that tree-isomorphism is contained in deterministic Logspace and thus certainly in (FO + TC+ $<$) = NL.

There is also a close connection between Dyn-FO and dynamic constant parallel time. It is well known that first-order queries can be performed in (uniform) AC^0, i.e., in constant parallel time on a CRCW-PRAM with a polynomial number of processors. Moreover, FO and constant parallel time correspond precisely when there are built-in ordering and arithmetic predicates available on the input structure ([Imm89b]). We do not assume such predicates are available in our Dyn-FO setting, but as part of our upper bound proof we show that ordering and arithmetic on the *activated domain* can actually be built up dynamically using first-order updates, thereby showing that Dyn-FO, even without ordering and arithmetic, corresponds precisely to dynamic AC^0. This provides some explanation as to why lower bounds for Dyn-FO are so difficult to obtain when the arity of the auxiliary relations is ≥ 2 ([DS97a]). Proving such a lower bound would amount to proving a lower bound on dynamic complexity in a fully endowed model of computation.

The sequential dynamic complexity of tree isomorphism was studied in [CN96] with applications to graph drawings, and it was shown there that dynamic tree isomorphism can be performed with updates that take poly-logarithmic time on a RAM. Our, inherently parallel, first-order dynamic setting is substantially different than their sequential dynamic setting, and the algorithms and data-structures used in [CN96] bear little relation to ours. It is interesting to note however that both the sequential and parallel dynamic setting apparently experience a particular bottleneck: the transitive closure query, TC (on directed graphs), is not known to have any better dynamic complexity than static complexity in either setting. In particular, TC is neither known to be contained in Dyn-FO, nor known to be updatable in polylogarithmic time on a RAM (see [KMW96] for some explanation of the sequential bottleneck).

The method we use to show tree isomorphism is in Dyn-FO is partly based on an idea of [EI95], where it was shown how to express tree-isomorphism in $(FO + COUNT)[\log n]$, a logic which allows logarithmic depth inductive defintions over (FO + COUNT) formulas. We show our result in two stages: first we use ideas similar to [EI95] to show how to perform tree isomorphism dynamically using $(FO + COUNT)$ updates, second, we show how the need for counting quantifiers (and numbers) can be removed by actually maintaining dynamically the necessary counting information (as well as building dynamically the needed relations to simulate numbers) using only first-order updates.

The paper is organized as follows: in section 2 we provide the background and definitions for Dyn-FO and the tree isomorphism problem; in section 3 we show how to perform tree isomorphism with $(FO + COUNT)$ updates; in section 4 we show how to remove the need for counting quantifiers (and numbers) by maintaining the necessary counting and arithmetic information dynamically using first-order updates, and we observe some consequences regarding why lower-bounds for Dyn-FO are so difficult; we make concluding remarks in section 5.

2 Background

We follow the definitions and presentation of Dyn-FO essentially as given by [PI97]. Since we use Dyn-$(FO +$ COUNT) at an intermediate step, we will use their more general definition of Dyn-$\mathcal{C}$ for an arbitrary logical lan-

putation can be any logic or complexity class. This way a hierarchy of dynamic complexity classes can be defined based on the strength of the update language.

guage $\mathcal{C}$. We will assume some familiarity with the basics of first-order logic and descriptive complexity (see [Imm98, Imm89a]).

For simplicity, let us focus on the setting where our input vocabulary σ is the vocabulary of graphs with one binary relation and two constants representing vertices, $\sigma = \langle E^2, c_1, c_2\rangle$. All our definitions easily generalize to an arbitrary input vocabulary σ. We will be maintaining auxiliary data using relations from a relational auxiliary vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}\rangle$. Let $S \subseteq \mathrm{STRUCT}[\sigma]$ be a set of structures from the vocabulary σ, in our case a set of *undirected* graphs. We say that S is a *property* of (undirected) graphs or an (undirected) graph *problem.* Given $A = \langle V^A, E^A, c_1^A, c_2^A\rangle \in \mathrm{STRUCT}[\sigma]$ and $B = \langle V^A, R_1^B, \ldots, R_r^B\rangle \in \mathrm{STRUCT}[\tau]$, where A and B share a common universe, let $\mathbf{A}\|\mathbf{B} \doteq \langle \mathbf{V^A}, \mathbf{E^A}, \mathbf{R_1^B}, \ldots, \mathbf{R_r^B}, \mathbf{c_1^A}, \mathbf{c_2^A}\rangle \in \mathrm{STRUCT}[\sigma\cup\tau]$.

In general, one could study dynamic complexity in the presence of any set of modification operations.[2] In this paper we will focus on the operations INSERT and DELETE as proposed by [DS93, PI97]. Since we are working on undirected graphs represented as binary relations, we insert and delete an edge $\{a, b\}$ in the form of two directed edges $\{(a, b), (b, a)\}$.

For a graph $G = \langle V^G, E^G, c_1^G, c_2^G\rangle$ let:
$\mathbf{INSERT}(G, a, b) \doteq \langle V^G, E^G \cup \{(a, b), (b, a)\}, c_1^G, c_2^G\rangle$
$\mathbf{DELETE}(G, a, b) \doteq \langle V^G, E^G \setminus \{(a, b), (b, a)\}, c_1^G, c_2^G\rangle$

Definition 2.1 *Given a logical language $\mathcal{C}$, a property $S \subseteq \mathrm{STRUCT}[\sigma]$ is in the class* $\mathbf{Dyn}_{\{\mathbf{INSERT},\mathbf{DELETE}\}}$*-$\mathcal{C}$*[3] *iff there is a property $W \subseteq \mathrm{STRUCT}[\tau]$ with W expressible in $\mathcal{C}$ and a function $f : \mathrm{STRUCT}[\sigma] \to \mathrm{STRUCT}[\tau]$ such that:*

1. *For any graph G, $G \in S \iff f(G) \in W$*
2. *For $i \le r$, there are formulas $\varphi_{R_i}^{insert}(x_1, \ldots, x_{a_i}, u, v)$ and $\varphi_{R_i}^{delete}(x_1, \ldots, x_{a_i}, u, v)$ over the vocabulary $\sigma\cup\tau$ such that the following two conditions hold for any graph G and any $(a, b) \in V^G \times V^G$:*
   (a) *$f(\mathrm{INSERT}(G, a, b)) = \langle V^G, \varphi_{R_1}^{insert}(\bar{x}, a, b), \ldots, \varphi_{R_r}^{insert}(\bar{x}, a, b)\rangle$, where each formula $\varphi_{R_i}^{insert}(x_1, \ldots, x_{a_i}, a, b)$ is interpreted over $G\|f(G)$, yielding a new a_i-ary relation, and they together yield a structure of type τ.*
   (b) *$f(\mathrm{DELETE}(G, a, b)) = \langle V^G, \varphi_{R_1}^{delete}(\bar{x}, a, b), \ldots, \varphi_{R_r}^{delete}(\bar{x}, a, b)\rangle$.*
3. *The initial auxiliary data, when the input structure is empty, is expressible in $\mathcal{C}$. In other words: there are $\mathcal{C}$-formulas ψ_i over the vocabulary σ such that $f(\emptyset) = \langle \psi_1, \ldots, \psi_r\rangle$, where $\emptyset$ denotes the empty σ-structure and the formulas ψ_i are interpreted over $\emptyset$.*[4]

∎

To help the reader grasp this definition, we now repeat an often-used simple example of a property in Dyn-FO.

Example 2.2 Consider dynamically maintaining the *parity* of the number of edges of an undirected graph under the operations INSERT and DELETE. Statically, determining the parity of an edge relation is known not to be expressible in first-order logic (not even with arbitrary built-in predicates, even in existential second-order monadic logic [Ajt83]).

Dynamically, the problem is trivial. Our input vocabulary is that of graphs $\sigma = \langle E^2\rangle$. Our auxiliary vocabulary τ needs to store just one bit of information, i.e., a 0-ary relation P, $\tau = \langle P^0\rangle$, where P will store the current parity of the edge set. Our update formulas are:[5]

$$\varphi_P^{insert}(u, v) = (E_{old}(u, v) \wedge P_{old}) \vee (\neg E_{old}(u, v) \wedge \neg P_{old})$$

$$\varphi_P^{delete}(u, v) = (E_{old}(u, v) \wedge \neg P_{old}) \vee (\neg E_{old}(u, v) \wedge P_{old})$$

Clearly, when an edge $\{a, b\}$ is added to a graph G_{old}, the new graph G_{new} will have parity $P_{new} \equiv \varphi_P^{insert}(a, b)$. Similarly, φ^{delete} properly updates the parity bit P when an edge $\{a, b\}$ is deleted.

Going by the definition of Dyn-FO, the property W of τ-structures here is simply the property $P = \mathtt{true}$. The equally trivial formula for initializing P to $f(\emptyset)$ is just $\psi^{init} = \mathtt{false}$ (we are checking for odd parity). ∎

Let **TreeIso** be the graph problem defined as the set of undirected forests $F = \langle V^F, E^F, c_1^F, c_2^F\rangle$ such that there is a (rooted) isomorphism between the trees rooted at c_1^F and c_2^F which maps c_1^F to c_2^F. *In this conference paper* we demand furthermore that c_1^F and c_2^F are *not connected*, i.e. the trees rooted at c_1^F and c_2^F are disjoint. This restriction is unnecessary for our theorem to hold, but greatly simplifies our proof. We note that the lower bound of [EI95] holds for this version of tree isomorphism, and thus we can still conclude corollary 4.4.

At an intermediate step in our proof that TreeIso $\in$ Dyn-FO, we will be using the logic (FO + COUNT) [IL90, CFI92, Ete95], first-order logic augmented with counting quantifiers. In our treatment of (FO+COUNT), the logic is equipped with quantifiers of the form:

$(\exists i\, x)\ \varphi(x)\quad\equiv\quad$ "There exist at least i elements x of the universe for which $\varphi(x)$ holds."

[2] Obviously, these operations should cause minor change to the input structure and/or should themselves be definable in a simple logic, otherwise considering dynamic complexity would make little sense. We will discuss this issue further in the conclusion.

[3] We will usually drop the subscript $\{INSERT, DELETE\}$ from $\mathrm{Dyn}_{\{INSERT,DELETE\}}$-$\mathcal{C}$.

[4] Note that, given the formulas ψ_i, we can modify the update formulas φ_i, so that they take care of the initial setting of the auxiliary data as a special case.

[5] The *old* subscripts above are provided only to emphasize the fact that we are making a transition from the old version of the input and auxiliary relations to their new version. The subscript is not actually part of our syntax.

where i is now a free variable that ranges over a separate number domain $\{1, \ldots, n\}$, which we assume exists and is equinumerous with our ordinary universe. On this number domain, we assume we have available the usual built-in arithmetic predicates $\{\leq, +, \times\}$. We will show, in the end, that neither counting nor numbers are necessary because we can maintain the necessary counting information dynamically, but it will be a convenient intermediate step in our proof.

3 Maintaining tree isomorphism using first-order + counting.

Theorem 3.1 TreeIso $\in$ Dyn-(FO + COUNT)

Proof

Definition 3.2 *Given a forest $F = (V, E)$, let* $\mathbf{path}(x, y)$ *be the reflexive transitive closure of the edge relation E. Let $\mathbf{st_x(y)}$ denote the subtree rooted at y in the tree rooted at x in F (in particular,* $\mathrm{path}(x, y)$ *must hold for the subtree to be defined). We write $\mathbf{st_x(x')} \cong \mathbf{st_y(y')}$ to mean there is a (rooted) isomorphism between $st_x(x')$ and $st_y(y')$ which maps x' to y' . Let* $\mathrm{dist}(x, y)$ *denote distance in the forest (∞ if $\neg\mathrm{path}(x, y)$), and let* $\mathbf{dist}(x, y, d)$ *be the graph of this function restricted to finite distances. Let* $\boldsymbol{onpath(x, y, z)}$ *mean "*$\mathrm{path}(x, y)$ *and z is on the* unique *shortest path from x to y". Let* $\mathbf{T\text{-}ISO(x, x', y, y')}$ *mean that the subtrees rooted at x' and y', in the trees rooted at x and y, respectively, are isomorphic and disjoint. In symbols* $\mathrm{T\text{-}ISO}(x, x', y, y')$ *means "*$\mathrm{path}(x, x') \wedge \mathrm{path}(y, y') \wedge \neg\mathrm{onpath}(x, y', x') \wedge \neg\mathrm{onpath}(y, x', y') \wedge (st_x(x') \cong st_y(y'))$*". Let*

$$\tau \doteq \langle \mathrm{path}, \mathrm{dist}, \mathrm{onpath}, \mathrm{T\text{-}ISO} \rangle$$

∎

We prove the theorem by maintaining the auxiliary relations in the vocabulary τ according to their described meaning. Let $W = \{B \in \mathrm{STRUCT}[\tau] \mid B \models \mathrm{T\text{-}ISO}(c_1, c_1, c_2, c_2)\}$. Clearly, $F \in$ TreeIso if and only if $f(F) \in W$. Therefore, it suffices to maintain τ.

Proposition 3.3 $\mathrm{dist}(x, y, d)$ *can be updated by an* FO *formula*[6] *after both* $\mathrm{INSERT}(F, a, b)$ *and* $\mathrm{DELETE}(F, a, b)$.

Proof Let

$$\begin{aligned} \psi \quad \equiv \quad & (\exists d_1, d_2\ (d = d_1 + d_2 + 1)\ \wedge \\ & ((\mathrm{dist}_{old}(x, a, d_1) \wedge \mathrm{dist}_{old}(b, y, d_2)) \vee \\ & (\mathrm{dist}_{old}(x, b, d_1) \wedge \mathrm{dist}_{old}(a, y, d_2)) \end{aligned}$$

The following simple formulas perform the updates.

$$\begin{aligned} \varphi_{dist}^{insert}(x, y, d, a, b) \quad \doteq \quad & \mathrm{dist}_{old}(x, y, d) \vee \\ & (\neg\exists d'\ \mathrm{dist}_{old}(x, y, d') \wedge \psi) \end{aligned}$$

$$\varphi_{dist}^{delete}(x, y, d, a, b) \doteq \mathrm{dist}_{old}(x, y, d) \wedge (\neg E_{old}(a, b) \vee \neg\psi)$$

∎

It follows that $\mathrm{path}(x, y)$ and $\mathrm{onpath}(x, y, z)$ can also be maintained with first-order formulas because

$$\begin{aligned} \mathrm{path}(x, y) \quad & \Leftrightarrow \quad \exists d\ \mathrm{dist}(x, y, d) \\ \mathrm{onpath}(x, y, z) \quad & \Leftrightarrow \quad \exists d_1, d_2, d \\ & \qquad (\mathrm{dist}(x, y, d) \wedge \mathrm{dist}(x, z, d_1) \wedge \\ & \qquad \mathrm{dist}(z, y, d_2) \wedge d_1 + d_2 = d) \end{aligned}$$

Lemma 3.4 T-ISO *can be updated after* $INSERT(F, a, b)$ *by an* (FO + COUNT) *formula.*

Proof We split our expression into subcases. Let

$$\begin{aligned} \gamma \quad \doteq \quad & \neg\mathrm{path}(x, x') \vee \neg\mathrm{path}(y, y') \vee \\ & \neg(\mathrm{onpath}(x, a, x') \vee \mathrm{onpath}(x, b, x') \vee \\ & \mathrm{onpath}(y, a, y') \vee \mathrm{onpath}(y, b, y')) \end{aligned}$$

The formula γ distinguishes two subcases. We will define formulas φ_1 and φ_2 for the two subcases, so that the update formula for T-ISO after an insertion is given by

$$\varphi_{\mathrm{T\text{-}ISO}}^{insert}(x, x', y, y', a, b) \equiv (\gamma \wedge\ \varphi_1) \vee (\neg\gamma \wedge\ \varphi_2)$$

Here are the subcases:

1. When γ holds, either (a) one of $st_x(x')$ or $st_y(y')$ don't exist, or (b) neither a nor b are in $st_x(x')$, nor in $st_y(y')$. In either case, there is nothing to change:

   $$\varphi_1(x, x', y, y', a, b) \doteq \mathrm{T\text{-}ISO}_{old}(x, x', y, y')$$

2. When $\neg\gamma$ holds, it must be that either (a) one of a or b is in one of $st_x(x')$ or $st_y(y')$, while neither is in the other, or (b) $st_x(x')$ and $st_y(y')$ will not be disjoint after the insertion, in which case, $\mathrm{T\text{-}ISO}_{new}$ must be `false`. We can again distinguish these two cases with another simple formula δ which holds iff case (a) holds. We let $\varphi_2 = \delta \wedge \varphi_2'$, and we are left with the task of writing a formula φ_2' which updates T-ISO when case (a) holds.

   To do this, we use an idea from [EI95] designed to distinguish two trees recursively, but here the recursion will not be necessary because of the auxiliary information. Assume, w.l.o.g.[7], that $a \in st_x(x')$. Then $st_x(x') \cong st_y(y')$ *if and only if* there is a point a_{dup} in $st_y(y')$, and a point b_{dup} which is a child of a_{dup} in $st_y(y')$ such that the following conditions hold:

   (a) $st_b(b) \cong st_y(b_{dub})$

   (b) $\exists\ d,\ \mathrm{dist}(x', a, d) \wedge \mathrm{dist}(y', a_{dup}, d)$

[6] in the presence of a number domain, which we later show is not necessary.

[7] Again, because we can distinguish the various cases with first-order formulas.

(c) $\forall\ d' \leq d$, let $y_{d'}$ be the vertex such that onpath$(y', a_{dup}, y_{d'})$ and dist$(y', y_{d'}, d')$. Let $x_{d'}$ be the counterpart of $y_{d'}$, such that onpath$(x', a, x_{d'})$ and dist$(x', x_{d'}, d')$. Then there are exactly the same number, j, of children z of $y_{d'}$ in $st_y(y')$ where $\neg$onpath(y', a_{dup}, z), as the number of children w of $x_{d'}$ such that $\neg$onpath(x', a, w). Moreover, for each such z there are exactly the same number, i of children z' of $y_{d'}$ such that $\neg$onpath(y', a_{dup}, z') and $st_{y_{d'}}(z) \cong st_{y_{d'}}(z')$, as the number of children w' of $x_{d'}$ such that $\neg$onpath(x', a, w') and $st_{y_{d'}}(z) \cong st_{x_{d'}}(w')$. (Please see figure 1 for a depiction of this situation.)

We will henceforth call the procedure just described *pivoting* at a and a_{dup}, and we will call the paths from x' to a and from y' to a_{dup} the *critical paths*. The key observation is that in the above characterization of isomorphic subtrees the only smaller subtrees that need to be checked recursively for isomorphism are those that do not contain the edge $\{a, b\}$. We can therefore use T-ISO$_{old}$ to determine isomorphism for those subtrees. Using this observation, it isn't difficult to see that the above characterization can be translated directly into a (FO + COUNT) formula φ_2' which uses the auxiliary relations and uses counting quantifiers to assert the number of children of a node whose subtrees are isomorphic to a given subtree.

∎

Lemma 3.5 T-ISO *can be updated after a* DELETE(F, a, b) *operation using an* (FO + COUNT) *formula.*

Proof Again, we deal with three cases separately. The third case is the most interesting.

1. Neither a nor b are in $st_x(x')$ or in $st_y(y')$.

   In this case $\varphi_1(x, x', y, y') \equiv$ T-ISO$_{old}(x, x', y, y')$

2. Both a and b are in one of $st_x(x')$ or $st_y(y')$, while neither is in the other.

   Suppose without loss of generality that a and b are in $st_x(x')$ and that a is closer to x' than b. Then, again using the characterization from [EI95], we *pivot* at a and at a point a point a_{dup} which we quantify in $st_y(y')$. The only thing to take care of is that the b subtree of a should not be considered in the counting, but this amounts to a simple modification of the pivoting procedure from Lemma 3.4.

3. Both a and b are in both $st_x(x')$ and $st_y(y')$. Furthermore, removing edge $\{a, b\}$ disconnects x' from y', and therefore makes the resulting subtrees disjoint. (Otherwise, the subtrees will not become disjoint, and T-ISO(x, x', y, y') will remain `false`.)

   This case is more interesting, as the two subtrees do not start disjoint. We will actually have to use the pivoting procedure of Lemma 3.4 *twice* in order to check isomorphism of the resulting subtrees in this case. It is worth pointing out here that this case will be dealt with in a manner similar to how we will, in the full version of this paper, remove the requirement that $st_x(x')$ and $st_y(y')$ are to be disjoint for T-ISO(x, x', y, y') to hold.

   Suppose, w.l.o.g., that a is closer to x' than to y', and that dist$(x', a) \geq$ dist(y', b). The path from x' to y' first goes through a before seeing b. We have to check whether $st_x(x')$ is isomorphic to $st_y(y')$ after we remove the edge $\{a, b\}$. We will apply the pivoting technique of Lemma 3.4 in two stages.

   We begin by pivoting at a with respect to x', and we quantify a_{dup} such that $a_{dup} \notin st_b(a)$. This insures that a_{dup} will reside in $st_y(y')$ after $\{a, b\}$ has been removed.

   Let dist$(x', a) = d =$ dist(y', a_{dup}). Let v_{dup} be the unique node such that onpath(y', a_{dup}, v_{dup}) and onpath(b, a_{dup}, v_{dup}) and onpath(b, y', v_{dup}). Let $d'' = dist(y', v_{dup})$. Let v be the node onpath(x', a, v) and $dist(x', v) = d''$. v_{dup} defines a key point on the the critical path from the pivot a_{dup} to the root y' of the subtree $st_y(y')$ which has to be dealt with separately because it is the vertex from which we can reach b, and hence the edge $\{a, b\}$ from the critical path. v is v_{dup}'s counterpart on the other critical path.

   We do the check for isomorphic subtrees at all levels on the critical path from the roots to the pivots, except that we deal with the check for v vs. v_{dup} separately. We first check that for each level d' on the critical path such that $d' \neq d$ and $d' \neq d''$, the condition on the count of isomorphic subtrees at that level holds, as specified in part (c), case (2), of Lemma 3.4.

   We are then left with the task of verifying the condition for v and v_{dup}. Let b^{top} be the child of v_{dup} in $st_{y'}(v_{dup})$ such that onpath(v_{dup}, b, b^{top}). If no such child exists then it must be that $v_{dup} = b$. In this case, it must be that $a = v$ and we are in a simplified situation: we can check that $st_x(x') \cong st_y(y')$, after deletion of $\{a, b\}$, by pivoting at b in $st_x(x')$ and a in $st_y(y')$, and checking that the conditions on the critical path hold for $d' \neq< d$, with respect to these two pivots.

   In case b^{top} does exist, we first check that v has a child b^{top}_{dup} such that $st_{v_{dup}}(b^{top}) \cong st_v(b^{top}_{dup})$, when the edge $\{a, b\}$ is removed from the forest. To do this, we *again* do pivoting, this time one pivot is at b in $st_{v_{dup}}(b^{top})$, i.e., the top node on the critical path is b^{top}, and for the other pivot we quantify b_{dup} in $st_v(b^{top}_{dup})$. We now check whether $st_{v_{dup}}(b^{top}) \cong st_v(b^{top}_{dup})$, after $\{a, b\}$ is removed, using the old T-ISO relation, because the edge $\{a, b\}$ does not to occur in any sub-subtrees which occur off of the critical paths from roots b^{top} and b^{top}_{dup} to pivots b and b_{dup}, respectively.

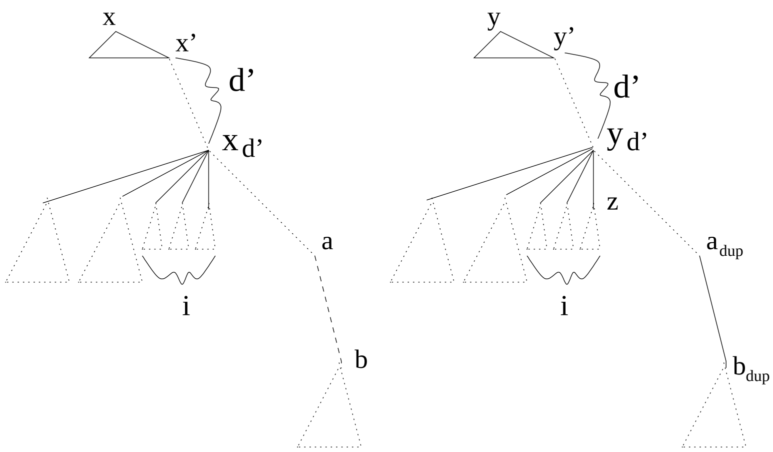

Figure 1: Characterization of isomorphic subtrees

Finally, we check that for each child s of v_{dup} in $st_{y'}(v_{dup})$, such that both $s \neq b^{top}$ and $\neg$onpath(v_{dup}, a_{dup}, s), the number of siblings s' of s, satisfying both these conditions, such that $st_{v_{dup}}(s) \cong st_{v_{dup}}(s')$ (which we can check using the T-ISO$_{old}$ again) is the same as the number of children t of v in $st_{x'}(v)$ such that $t \neq b^{top}_{dup}$ and $\neg$onpath(v, a, t) such that $st_{v_{dup}}(s) \cong st_v(t)$ (which we can again check using T-ISO$_{old}$). This rather long-winded description finishes all requirements for checking isomorphism of $st_x(x')$ and $st_y(y')$ after deletion of $\{a, b\}$. One can indeed convert this entire description to an (FO + COUNT) formula, but we will spare the reader, who has patiently read this far, all such detail.

∎

That concludes the proof that TreeIso $\in$ Dyn-FO. ∎

We note here in passing that [LW97] used an SQL-like language, which incorporates a certain form of counting, to incrementally update various graph properties such as alternating reachability (which is P-complete). However, their formalism essentially allows for counting up to very large numbers, exponentially larger than the universe, whereas we we are limited to counting only up to the size of the universe. Hence, our result is incompatible with theirs. Indeed, we show in the next section that we can completely eliminate our need for counting and numbers.

4 Removing the need for counting and numbers

We now show how to remove the need for counting quantifiers as well as the need for numbers and arithmetic.

The first surprising fact is that we can eliminate the need for numbers altogether. The reason is that the only arithmetic we need is *on the activated domain* and we can build the needed arithmetic predicates on this domain as auxiliary relations. Consider a sequence of insertions and deletions of edges. Let the **activated domain** (ad) of a graph G be those vertices that have at some point during this sequence of insertions and deletions had an edge incident to them. Let $ad(x)$ be the predicate that defines this set. Let $\leq_{ad}(\mathbf{x}, \mathbf{y})$ denote a total ordering on the activated domain, defined by the order in which vertices entered the activated domain. To resolve ties, when INSERT(G, a, b) occurs, $a \neq b$ and $a, b \notin ad$, we arbitrarily choose to let vertex a into the activated domain prior to vertex b.[8] Once we have an ordering $\leq_{ad}$, we can view ad as a set of numbers $\{1, \ldots, n_{ad}\}$. Let $\max_{ad}(x)$ be the predicate that hold only for the current maximum, n_{ad}. Let $\mathbf{BIT}_{ad}(\mathbf{x}, \mathbf{i})$ denote the bit predicate on these numbers, meaning (when $i, x \in ad$ are viewed according to their representation as binary numbers) "the i'th bit of the number x is on" (we assume bit positions are numbered starting at 1). Let $+_{ad}(x, y, z)$ and $\times_{ad}(x, y, z)$ be graphs of the ordinary addition and multiplication function on the domain ad.

Lemma 4.1 *We can build and maintain auxiliary relations that define* $\mathrm{ad}(x)$, $\leq_{\mathrm{ad}}(x, y)$, $\max_{\mathrm{ad}}(x)$, $\mathrm{BIT}_{\mathrm{ad}}(x, i)$, $+_{\mathrm{ad}}(x, y, z)$ *and* $\times_{\mathrm{ad}}(x, y, z)$.

Proof All these relations do not change after a delete operation, so we only write formulas for updates after an insertion.

$$
\begin{aligned}
\varphi^{insert}_{ad}(x, a, b) &\equiv ad_{old}(x) \vee x = a \vee x = b \\
\varphi^{insert}_{\leq_{ad}}(x, y, a, b) &\equiv \leq_{ad_{old}}(x, y) \vee (\varphi^{insert}_{ad}(y, a, b) \wedge \\
&\quad (x = y \vee ad_{old}(x))) \vee \\
&\quad (\varphi^{insert}_{ad}(x, a, b) \wedge \varphi^{insert}_{ad}(y, a, b) \wedge \\
&\quad x = a \wedge (y = a \vee y = b)) \\
\varphi^{insert}_{\max_{ad}}(x, a, b) &\equiv \varphi^{insert}_{ad}(x, a, b) \wedge
\end{aligned}
$$

[8] Note that, even though undirected edges are being inserted, we are using the inherent asymmetry of INSERT(G, a, b) in a crucial. This is justified since any realistic way to actually specify an unordered pair of vertices must provide the vertices is some order.

$$\forall z(\varphi_{ad}^{insert}(z,a,b) \rightarrow \varphi_{\leq_{ad}}^{insert}(z,x,a,b))$$

To express $\mathrm{BIT}_{ad}(x,i)$, observe that the only interesting situation is when x is new to ad. Using the fact that *addition* is first-order expressible, we can identify the prior *maximum* element, z, of ad and increment its binary representation by 1 or 2, depending on whether x is the first or second new entry into ad.

$\varphi_{\mathrm{BIT}_{ad}}^{insert}(x,i,a,b) \equiv$
$\mathrm{BIT}_{ad_{old}}(x,i) \vee ((\neg ad_{old}(x) \wedge (x = a \vee (ad_{old}(a) \wedge x = b)) \wedge$
$\exists z(\max_{old}(z) \wedge \exists y(ad(y) \wedge \neg \mathrm{BIT}_{old}(z,y) \wedge \forall j(j <_{ad} y \rightarrow \mathrm{BIT}_{old}(z,j)) \wedge ((i > y \wedge \mathrm{BIT}_{old}(z,i)) \vee i = y))) \vee$
(increment the binary representation by 2, similarly)

The maintainability of $+_{ad}(x,y,z)$ and $\times_{ad}(x,y,z)$ follows from the known fact from [BIS90] that $+$ and $\times$ are first-order expressible in terms of $\leq$ and BIT. ∎

[DS97b] defines a weaker formalism, named deterministic-FOIES, in which the auxiliary data must be uniquely determined by the current input, independent of the history of insertions and deletions. Note that in building our arithmetic predicates we are explicitly using the ordered history of insertions to order our domain. Thus, our elimination of numbers doesn't work in their weaker formalism.

Now, to remove counting quantifiers, we will maintain the following auxiliary relation which actually maintains the necessary counting information for us. Let **sibling-iso-count**$(\mathbf{x}, \mathbf{x}', \mathbf{y}, \mathbf{y}', \mathbf{y}'', \mathbf{d})$ denote: "path(x,x') and $y \in st_x(x')$, and the number of siblings s of y in $st_x(x')$ which are not equivalent to y' or y'' and for which $st_x(s) \cong st_x(y)$ is d". Note that the relation `sibling-iso-count`(x,x',y,y',y'',d) only holds when $ad(d)$, and thus we only need arithmetic on numbers up to $|ad|$.

Lemma 4.2 `sibling-iso-count` *can be updated with first-order formulas.*

Proof *(very brief sketch)* The main point is that after an INSERT (or DELETE) either:

1. The edge $\{a,b\}$ is in one of the sibling subtrees. In this case, the count d at most changes by one, and we can tell what this change is by using the subtree isomorphism formula, using `sibling-iso-count`$_{old}$ (because the counting required to check the subtrees for isomorphism will not change because the edge $\{a,b\}$ does not appear in anything we are counting recursively).

2. The edge $\{a,b\}$ is added in the subtree $st_x(y)$ itself. In this case we can check if the new tree is isomorphic to one of its siblings $st_x(s)$, and we obtain the number of such isomorphic siblings by checking the old count of siblings isomorphic to $st_x(s)$ using `sibling-iso-count`$_{old}$.

∎

Substituting the use of `sibling-iso-count` in the counting formula for updating T-ISO, we obtain:

Theorem 4.3 TreeIso $\in$ Dyn-FO

Corollary 4.4 Dyn-FO $\not\subseteq$ (FO + TC + COUNT)

Proof Follows immediately from Theorem 4.3 and the lower bound of [EI95]. ∎

Why are lower-bounds for Dyn-FO so difficult?

Lemma 4.1 sheds some light on why lower bound proofs for Dyn-FO are so hard. Let Dyn-(uniform-)AC^0 be the class of dynamic problems where it is possible to update both the query and the auxiliary data with a Dlogtime-uniform constant-depth, polynomial-size boolean circuit with unbounded fan-in, where the amount of auxiliary data is polynomially long in the length of the input string, and the length of the input string is fixed. (For a formal and more general definition of Dyn-$\mathcal{C}$, for any complexity class $\mathcal{C}$, please see [PI97].)

Let Dyn-FO$^{\$}$ be the same as Dyn-FO except that we assume the entire universe is activated. This could be enforced by, e.g., assuming that starting from the $\emptyset$ structure, an edge incident to every vertex is inserted and deleted from the graph (we omit a formal definition).

Corollary 4.5 Dyn-FO$^{\$}$ = Dyn-AC^0

Proof In [BIS90] it was shown that *uniform*-AC^0 is equivalent to FO$(\leq, \mathrm{BIT})$, first-order logic equipped with the given built-in arithmetic predicates. Once the entire universe is activated, by Lemma 4.1 we can build all the necessary arithmetic predicates on the universe dynamically. ∎

Dong and Su [DS97a] considered lower-bounds for Dyn-FO when the *arity* of auxiliary relations is bounded by some fixed k. They were able to use EF games to obtain lower-bounds for graph properties when the arity was restricted to 1, but for arity ≥ 2, they were only able to prove weak lower-bounds requiring higher arity in the input relations, and they relied heavily on circuit lower-bounds of [Ajt83, Cai90]. In light of Lemma 4.1, the difficulty of proving lower-bounds with fixed input arity and large auxiliary data is not surprising. Let Dyn-FO$_k$ denote Dyn-FO with the arity of auxiliary relations restricted to k, and let Dyn-$\mathrm{AC}^0[n^k]$ denote Dyn-AC^0 with only $O(n^k)$ auxiliary bits. It follows from the fact ([BIS90]) that $\leq$ and BIT are sufficient to define all the arithmetic predicates that:

Corollary 4.6 *For* $k \geq 2$, Dyn-FO$_k^{\$}$ = Dyn-$\mathrm{AC}^0[n^k]$.

Thus, proving lower-bounds for Dyn-FO with auxiliary arity ≥ 2 amounts to proving dynamic lower bounds in a general model of computation. Such lower bounds have indeed been *very* difficult to obtain. In fact there is no known lower bound for even the extremely weak update model of logarithmic depth decision trees ([Mil97]), which are even weaker than depth 2 circuits.

5 Conclusion and Discussion

Dynamic complexity is a new area where the proper mathematical models are not yet clear. Dyn-FO ([DS93, PI97]) is a model for dynamic query evaluation based on first-order logic as a query/update language.

We have shown that dynamic tree isomorphism is contained in Dyn-FO. Various other interesting graph problems are already known to be contained in Dyn-FO ([PI97, DS95]). However, it appears that Dyn-FO is incompatible with standard complexity classes and logics. In particular, it follows from our upper bound, and the lower bound of [EI95], that Dyn-FO $\not\subseteq$ (FO + TC + COUNT), but we as yet don't know whether TC, the transitive closure query (over arbitrary directed graphs), is itself in Dyn-FO.

There has been little success in proving strong lower bounds for Dyn-FO. We've provided some explanation for why this is so by showing that Dyn-FO can build all the arithmetic predicates on the activated subset of its universe and, once the entire universe is activated, is as powerful as dynamic AC^0.

Is Dyn-FO a realistic model for dynamic query evaluation? I feel one particular weakness of the model, as it is usually formulated, is that it seems unreasonable to assume that modifications to the database come only in the form of insertion or deletion of tuples. In a realistic setting, a database user may be making a variety of minor modifications which don't necessary change only a constant number of tuples. In particular, these modifications might themselves be specified as formulas in some restricted logic over the input vocabulary. Note that, e.g., INSERT(G, a, b) and DELETE(G, a, b) can trivially be described as formulas over $\sigma = \langle E \rangle$ parameterized by a and b.

The question then arises: *Is it possible to update interesting queries when modifications to the input are specified as (any fixed set of) formulas in some restricted language*? More concretely: *Can we update queries like parity, connectivity, tree isomorphism, etc., in Dyn*$_\Delta$*-FO, where* Δ *is any finite set of operations specified by (first-order) formulas over the input vocabulary? What if* Δ *can consist only of formulas from a restricted subset of first-order logic, such as quantifier-free formulas?*

Acknowledgment. Thanks to Leonid Libkin and Peter Bro Miltersen for valuable comments and discussions. Thanks also to an anonymous referee for two very helpful corrections.

References

[Ajt83] M. Ajtai. Σ^1_1-formulae on finite structures. *Ann. Pure and Appl. Logic*, 24:1–48, 1983.

[BIS90] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *JCSS*, 41:274–306, 1990.

[BLT86] J. Blakeley, P. Larson, and F. Tompa. Efficiently updating materialized views. In *Proc. ACM SIGMOD*, pages 61–71, 1986.

[Cai90] J. Y. Cai. Lower bounds for constant-depth circuits in the presence of help bits. *Information Processing Letters*, 36:79–83, October 1990.

[CFI92] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, **12** (4):389–410, 1992.

[CN96] S. W. Cheng and M. P. Ng. Isomorphism testing and display of symmetries in dynamic trees. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 202–211, 1996.

[DS93] G. Dong and J. Su. First-order incremental evaluation of datalog queries. In *Proc. of 4th International Workshop on Database Programming Languages, C. Beeri, A. Ohori and D. Shasha (Eds)*, pages 295–308. Workshop in Computing Series, Springer-Verlag, 1993.

[DS95] G. Dong and J. Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*, 120:101–106, 1995.

[DS97a] G. Dong and J. Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. to appear in *Journal of Computer and System Sciences*, 1997. Conference version appeared as "Space-Bounded FOIES" in *PODS'95*.

[DS97b] G. Dong and J. Su. Deterministic foies are strictly weaker, 1997. to appear in *Annals of Mathematics and Artificial Intelligence*.

[EI95] K. Etessami and N. Immerman. Tree canonization and transitive closure. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 331–341, 1995.

[Ete95] K. Etessami. Counting quantifiers, successor relations, and logarithmic space. In *10th Structure in Complexity Theory Conf.*, 1995.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. ACM SIGMOD*, 1995.

[GM95] A. Gupta and I. Mumick. Maintenance of materialized views: problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–19, 1995.

[GMS93] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. ACM SIGMOD*, pages 157–166, 1993.

[IL90] N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In Alan Selman, editor, *Complexity Theory Retrospective*, pages 59–81. Springer-Verlag, 1990.

[Imm89a] N. Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory*, volume 38, pages 75–91. Proc. Symp. in Applied Math., American Mathematical Society, 1989.

[Imm89b] N. Immerman. Expressibility and parallel complexity. *SIAM J. of Comput.*, 18:625–638, 1989.

[Imm98] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1998. to appear.

[KMW96] S. Khanna, R. Motwani, and R. Wilson. On certificates and lookahead in dynamic graph problems. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, pages 222–231, 1996.

[Lin92] S. Lindell. A logspace algorithm for tree canonization. In *24th Symp. on Theory of Comput.*, pages 400–404, 1992.

[LW97] L. Libkin and L. Wong. Incremental recomputation of recursive queries with nested sets and aggregate functions. In *Proceeding of DBPL*, 1997.

[Mil97] P. Miltersen. personal communication, 1997.

[MSVT94] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1 August 1994.

[PI97] S. Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, 1997. Conference version appeared in PODS'94.

[QW91] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3:337–341, 1991.

[Via97] V. Vianu. Databases and finite model theory. In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity Theory*, volume 31 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.