Reachability is in DynFO

Samir Datta[1], Raghav Kulkarni[2], Anish Mukherjee[1], Thomas Schwentick[3], and Thomas Zeume[3]

[1] Chennai Mathematical Institute, India, (sdatta,anish)@cmi.ac.in
[2] Center for Quantum Technologies, Singapore, kulraghav@gmail.com
[3] TU Dortmund University, Germany,
(thomas.schwentick,thomas.zeume)@tu-dortmund.de

Abstract. We consider the *dynamic complexity* of some central graph problems such as Reachability and Matching and linear algebraic problems such as Rank and Inverse. As elementary change operations we allow insertion and deletion of edges of a graph and the modification of a single entry in a matrix, and we are interested in the complexity of maintaining a property or query. Our main results are as follows:

1. Reachability is in DynFO;
2. Rank of a matrix is in DynFO(+,×);
3. Maximum Matching (decision) is in non-uniform DynFO.

Here, DynFO allows updates of the auxiliary data structure defined in first-order logic, DynFO(+,×) additionally has arithmetics at initialization time and non-uniform DynFO allows arbitrary auxiliary data at initialization time. Alternatively, DynFO(+,×) and non-uniform DynFO allow updates by uniform and non-uniform families of poly-size, bounded-depth circuits, respectively.

The first result confirms a two decade old conjecture of Patnaik and Immerman [16]. The proofs rely mainly on elementary Linear Algebra. The second result can also be concluded from [7].

1 Introduction

Dynamic Complexity Theory studies dynamic problems from the point of view of Descriptive Complexity (see [13]). It has its roots in theoretical investigations of the view update problem for relational databases. In a nutshell, it investigates the logical complexity of updating the result of a query under deletion or insertion of tuples into a database.

As an example, the Reachability query asks whether in a directed graph there is a path from a distinguished node s to a node t. The correct result of this query (i.e., whether such a path exists in the current graph) can be maintained for *acyclic graphs* with the help of an auxiliary binary relation that is updated by a first-order formula after each insertion or deletion of an edge. In fact, one can simply maintain the transitive closure of the edge relation. In terms of Dynamic Complexity, we get that Acyclic Reachability is in DynFO. In this setting, a sequence of change operations is applied to a graph with a fixed set of nodes whose edge set is initially empty [16].

Studying first-order logic as an update language in a dynamic setting is interesting for (at least) two reasons. In the context of relational databases, first-order logic is a natural update language as such updates can also be expressed in SQL. On the other hand, first-order logic also corresponds to circuit-based low level complexity classes; and therefore queries maintainable by first-order updates can be evaluated in a highly parallel fashion in dynamic contexts.

We also consider two extensions, DynFO$(+,\times)$ and non-uniform DynFO, whose programs can assume at initialization time multiplication and addition relations on the underlying universe of the graph, and arbitrarily pre-computed auxiliary relations, respectively. These two classes contain those problems that can be maintained by uniform and non-uniform families of poly-size, bounded-depth circuits, respectively.

The Reachability query is of particular interest here, as it is one of the simplest queries that can not be expressed (statically) in first-order logic, but rather requires recursion. Actually, it is in a sense prototypical due to its correspondence to transitive closure logic. The question whether the Reachability query can be maintained by first-order update formulas has been considered as one of the central open questions in Dynamic Complexity. It has been studied for several restricted graph classes and variants of DynFO [3,5,8,9,16,20]. In this paper, we confirm the conjecture of Patnaik and Immerman [16] that the Reachability query for general directed graphs is indeed in DynFO.

Theorem 1. *Directed Reachability is in* DynFO.

Our main tool is an update program (i.e., a collection of update formulas) for maintaining the rank of a matrix over finite fields $\mathbb{Z}_p$ against updates to individual entries of the matrix. The underlying algorithm works for matrix entries from arbitrary integer ranges, however, the corresponding DynFO update program assumes that only small numbers occur.[4]

Theorem 2. *Rank of a matrix is in* DynFO$(+,\times)$.

Theorem 1 follows from Theorem 2 by a simple reduction. Whether there is a path from s to t can be reduced to the question whether some (i,j)-entry of the inverse of a certain matrix has a non-zero value, which in turn can be reduced to a question about the rank of some matrix. This reduction (and similarly those mentioned below) is very restricted in the sense that a single change in the graph induces only a bounded number of changes in the matrix. We further use the observation that for domain independent queries as the Reachability query, DynFO is as powerful as DynFO$(+,\times)$. The combination of these ideas resolves the Patnaik-Immerman conjecture in a surprisingly elementary way.

By reductions to Reachability it further follows that Satisfiability of 2-CNF formulas and regular path queries for graph databases can be maintained in DynFO. By another reduction to the matrix rank problem, we show that the

[4] More precisely, it allows only integers whose absolute value is at most the possible number of rows and columns of the matrix.

existence of a perfect matching and the size of a maximum matching can be maintained in non-uniform DynFO.

Theorem 3. PerfectMatching *and* MaxMatching *are in non-uniform* DynFO.

Related work Partial progress on the Patnaik-Immerman conjecture was achieved by Hesse [9], who showed that directed reachability can be maintained with first-order updates augmented with counting quantifiers, i.e., logical versions of uniform TC^0. More recently, Datta, Hesse and Kulkarni [3] studied the problem in the *non-uniform* setting and showed that it can in fact be maintained in non-uniform $\mathsf{AC}^0[\oplus]$, i.e., non-uniform DynFO extended by parity quantifiers.

Dynamic algorithms for algebraic problems have been studied in [17,18,19]. The usefulness of matrix rank for graph problems in a logical framework has been demonstrated in [14]. Both [18,14] contain reductions from Reachability to matrix rank (different from ours). A dynamic algorithm for matrix rank, based on maintaining a reduced row echelon form, is presented in [7]. This algorithm can also be used to show that matrix rank is in $\mathsf{DynFO}(+,\times)$. More details are discussed in Section 3.1.

In [18,19] a reduction from maximum matching to matrix rank has been used to construct a dynamic algorithm for maximum matching. While in this construction the inverse of the input matrix is maintained using Schwartz Zippel Lemma, we use the Isolation Lemma of Mulmuley, Vazirani and Vazirani's [15] to construct non-uniform dynamic circuits for maximum matching.

The question whether Reachability can be maintained by formulas from first-order logic has also been asked in the slightly different framework of First-Order Incremental Evaluation Systems (FOIES) [4]. It is possible to adapt our update programs to show that Reachability can be maintained by FOIES.

Organization After some preliminaries in Section 2, we describe in Section 3 dynamic algorithms for matrix rank and Reachability, independent of a particular dynamic formalism. In Section 4 we show how these algorithms can be implemented as DynFO programs. Section 5 contains open ends.

2 Preliminaries

We refer the reader to any standard text for an introduction to linear algebraic concepts (see, e.g., [2]). We briefly survey some relevant ones here. Apart from the concept of *vector space* use its *basis* i.e. a linearly independent set of vectors whose linear combination spans the entire vector space and its *dimension* i.e. the cardinality of any basis. We will use matrices as linear transformations. Thus an $n \times m$ matrix M over a field $\mathbb{F}$ yields a transformation $T_M : \mathbb{F}^m \to \mathbb{F}^n$ defined by $T_M : x \mapsto Mx$. We will abuse notation to write M for both the matrix and the transformation T_M. The *kernel* of M is the subspace of $\mathbb{F}^m$ consisting of vectors x satisfying $Mx = \mathbf{0}$ where $\mathbf{0} \in \mathbb{F}^n$ is the vector of all zeroes.

In this paper we mainly study the following algorithmic problems.

MatrixRank	Reach
Given: Integer matrix A Output: rank(A) over $\mathbb{Q}$	Given: Directed graph G, nodes s, t Question: Is there a path from s to t in G?
PerfectMatching	**MaxMatching**
Given: Undirected graph G Question: Is there a perfect matching in G?	Given: Undirected graph G Output: Maximum size of a matching in G

For each natural number n, $[n]$ denotes $\{1, \ldots, n\}$.

3 Dynamic algorithms for Rank, Reachability and others

In this section, we present dynamic algorithms in an informal algorithmic framework. Their implementation as dynamic programs in the sense of Dynamic Complexity will be discussed in the next section. However, the reader will easily verify that these algorithms are highly parallelizable (in the sense of constant time parallel RAMs or the complexity class AC^0). We first describe how to maintain the rank of a matrix. Then we describe how to maintain an entry of the inverse of a matrix by a reduction to the rank of a matrix, and show that this immediately yields an algorithm for Reachability in directed graphs.

Theorem 3 uses the algorithm for rank in combination with the Isolation Lemma [15], more specifically its use described in [1], and a reduction from maximum matching to rank [11].

3.1 Maintaining the rank of a matrix

In this subsection we show that the rank of a matrix A can be maintained dynamically in a highly parallel fashion. We describe the algorithm for integer matrices, but it can be easily adapted for matrices with rational entries. At initialization time, the algorithm gets a number n of rows, a number m of columns, and a bound N for the absolute value of entries of the matrix A. Initially, all entries a_{ij} have value 0. Each change operation changes one entry of the matrix.

First, we argue that for maintaining the rank of A it suffices to maintain the rank of the matrix (A mod p) for polynomially many primes of size $O(\max(n, \log N)^3)$. To this end recall that A has rank at least k if and only if A has a $k \times k$-submatrix A' whose determinant is non-zero. The value of this determinant is bounded by $n!N^n$, an integer with $O(n(\log n + \log N))$ many bits. Therefore, it is divisible by at most $O(n(\log n + \log N))$ many primes. By the Prime Number Theorem, there are $\sim \frac{\max(n, \log N)^3}{\log \max(n, \log N)^3}$ many primes in $[\max(n, \log N)^3]$. Hence for n large enough, the determinant of A' is non-zero if and only if there is a prime $p \in [\max(n, \log N)^3]$ such that the determinant of (A' mod p) is non-zero. Hence the rank of A is at least k if and only if there is a prime p such that the rank of (A mod p) is at least k. Thus in order to compute the rank of A it suffices to compute the rank of (A mod p) in parallel for the primes in $[\max(n, \log N)^3]$, and to take maximum over all such ranks.

$$\underset{\text{Matrix } A}{\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}} \bullet \underset{\text{Basis } B}{\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}} = \underset{A \cdot B}{\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 0 & 0 & \boxed{1} \end{pmatrix}}$$

Fig. 1. A basis A with an A-good basis B. The first three (column) vectors of B are in the kernel K. The principal components of the two other vectors are marked in red.

Now we show how to maintain the rank of a $n \times m$ matrix A over $\mathbb{Z}_p$. The idea is to maintain a basis of the column space that contains a basis of the kernel of A. The number of non-kernel vectors in the basis determines the rank of A.

By K we denote the kernel of A, i.e., the vector space of vectors v with $Av = 0$. For a vector v in $\mathbb{Z}_p^m$, we write $S(v)$ for the set of non-zero coordinates of Av, that is, the set of all i, for which $(Av)_i \neq 0$.

As auxiliary data structure, we maintain a basis B of $\mathbb{Z}_p^m$ with the following additional property, called *A-good*. A vector $v \in B$ is *i-unique* with respect to B and A, for some $i \in [n]$, if $i \in S(v)$ but $i \notin S(w)$, for every other $w \in B$. We omit A when it is clear from the context. A basis B of $\mathbb{Z}_p^m$ is *A-good* if every $v \in B - K$ is i-unique with respect to B and A, for some i. For $v \in B - K$ in an A-good basis B, the minimum i for which v is i-unique is called the *principal component* of v, denoted by $\mathrm{pc}(v)$. Figure 1 illustrates an A-good basis.

The following proposition shows that it suffices to maintain A-good bases in order to maintain matrix rank modulo p.

Proposition 1. *Let A be an $n \times m$ matrix over $\mathbb{Z}_p$ and B an A-good basis of $\mathbb{Z}_p^m$. Then rank$(A) = n - |B \cap K|$.*

We now show how to maintain A-good bases modulo a prime p. Initially, the matrix A is all zero and every basis B of $\mathbb{Z}_p^m$ is A-good, as all its vectors are in K. Besides B, the algorithm also maintains the vector Av, for every $v \in B$, which is easy to do, as each change affects only one entry of A.

It is sufficient to describe how the basis can be adapted when one matrix entry a_{ij} of A is changed. We denote the new matrix by A', its entries by a'_{ij}, its kernel by K' and, for a vector v, the set of non-zero coordinates of $A'v$ by $S'(v)$. Clearly, for every vector v, Av and $A'v$ can only differ in the i-th coordinate as the only difference between A and A' is that $a_{ij} \neq a'_{ij}$. Therefore, if the A-good basis B is not A'-good, this can be only due to changes of the sets $S'(v)$ with respect to i. More specifically,

(a) there might be more than one vector $v \in B$ with $i \in S'(v)$, and
(b) there might be a vector $u \in B$ such that $\mathrm{pc}(u) = i$ but $i \notin S'(u)$.

When constructing an A'-good basis B' from the A-good basis B, those two issues have to be dealt with. To state the algorithm, the following definitions are useful. Let u denote the unique vector from B with $\mathrm{pc}(u) = i$, if such a vector exists. The set of vectors $v \in B$ with $i \in S'(v)$ can be partioned into three sets U, V and W where

Algorithm 1 Computation of B' from B.

(0) Copy all vectors from B to B'
(1) If $U \cup V \neq \emptyset$ then:
  (a) Choose $\hat{v}$ as follows:
    (i) If $V \neq \emptyset$, let $\hat{v}$ be the minimal element in V (with respect to the lexicographic order obtained from the order on V).
    (ii) If $V = \emptyset$ and $U \neq \emptyset$, let $\hat{v} \stackrel{\text{def}}{=} u$.
  (b) Make $\hat{v}$ i-unique by the following replacements in B':
    (i) Replace each element $w \in W$ by $w - (A'w)_i (A'\hat{v})_i^{-1} \hat{v}$.
    (ii) If $\hat{v} \in V$, replace each element $v \in V$, $v \neq \hat{v}$, by $v - (A'v)_i (A'\hat{v})_i^{-1} \hat{v}$.
    (iii) If $\hat{v} \in V$ and $U \neq \emptyset$, replace u by $\hat{u} \stackrel{\text{def}}{=} u - (A'u)_i (A'\hat{v})_i^{-1} \hat{v}$.
  (c) If u exists and $i \notin S'(u)$ (note: $U = \emptyset$) then let $\hat{u} \stackrel{\text{def}}{=} u$.
(2) If $\hat{u}$ has been defined (note: $i \notin S'(\hat{u})$) and $S'(\hat{u}) \neq \emptyset$ then:
  (a) Choose k minimal in $S'(\hat{u})$.
  (b) Make $\hat{u}$ k-unique by replacing every vector $v \in B'$ with $k \in S'(v)$ by $v - (A'v)_k (A'\hat{u})_k^{-1} \hat{u}$.
(3) Compute $A'v$, for every $v \in B'$ (with the help of the vectors Au, for $u \in B$)

- $U = \{u\}$ if $i \in S'(u)$, otherwise $U = \emptyset$.
- V is the set of vectors $v \in B \cap K$ with $i \in S'(v)$; and
- W is the set of vectors $w \in B - K$, with $i \in S'(w)$ but $w \neq u$ (thus, in particular $\mathrm{pc}(w) \neq i$).

For vectors $v \in V$, only i is a candidate for being the principal component since $S'(v) = \{i\}$ for such v because $Av = 0$ and the vectors Av and $A'v$ may only differ in the i-th component.

The idea for the construction of the basis B' is to apply modifications to B in two phases. In the first phase, when $U \cup V \neq \emptyset$, a vector $\hat{v} \in U \cup V$ is chosen as the new vector with principal component i. The i-uniqueness of $\hat{v}$ is ensured by replacing all other vectors x with $i \in S'(x)$ by $x - (A'x)_i (A'\hat{v})_i^{-1} \hat{v}$, where $(A'\hat{v})_i^{-1}$ denotes the inverse of the i-th entry of $A'\hat{v}$. The second phase assigns, when necessary, a new principal component k to the vector u or to its replacement from the first phase. Furthermore it ensures the k-uniqueness of this vector. The detailed construction of B' from B is spelled out in Algorithm 1.

Proposition 2. *Let A and A' be $n \times m$ matrices such that A' only differs from A in one entry $a'_{ij} \neq a_{ij}$. If B is an A-good basis of $\mathbb{Z}_p^m$ and B' is constructed according to Algorithm 1 then B' is an A'-good basis of $\mathbb{Z}_p^m$.*

An anonymous referee pointed out that the above stated algorithm for matrix rank (modulo p) is very similar to a dynamic algorithm for matrix rank presented as Algorithm 1 in [7] in a context where parallel complexity was not considered. Indeed, both algorithms essentially maintain Gaussian elimination, but the algorithm in [7] maintains a stronger normal form (reduced row echelon form) that differs by multiplication by a permutation matrix from our form.

However, Algorithm 1 in [7], restricted to single entry changes and integers modulo p, can be turned into an AC^0 algorithm by observing that the sorting step 12 only requires moving two rows to the appropriate places.

3.2 Maintaining Reachability

Next, we give a dynamic algorithm for Reachability. To this end, we first show how to reduce Reachability to the test whether an entry of the inverse of an invertible matrix equals some small number. Testing such a property will in turn be reduced to matrix rank.

We remind the reader, that for Reachability the number n of nodes is fixed at initialization time and the edge set is initially empty. Afterwards in each step one edge can be deleted or inserted. For simplicity, we assume that two nodes s and t are fixed at initialization time and we are always interested in whether there is a path from s to t. To maintain Reachability for arbitrary pairs, the algorithm can be run in parallel, for each pair of nodes.

For a given directed graph $G = (V, E)$ with $|V| = n$, we define its adjacency matrix $A = A_G$, where $A_{u,v} = 1$ if $u \neq v$ and there is a directed edge $(u, v) \in E$, and otherwise $A_{u,v} = 0$.

The matrix $I - \frac{1}{n}A$ is strictly diagonally dominant, therefore it is invertible (see e.g. [12, Theorem 6.1.10.]) and its inverse can be expressed by its Neumann series as $(I - \frac{1}{n}A)^{-1} = I + \sum_{i=1}^{\infty}(\frac{1}{n}A)^i$. The crucial observation is that the (s, t)-entry of the matrix on the right-hand side is non-zero if and only if there is a directed path from s to t. Therefore it suffices to maintain $(I - \frac{1}{n}A)^{-1}$ in order to maintain Reachability. To be able to work with integers, we consider the matrix $B \stackrel{\text{def}}{=} nI - A$ rather than $I - \frac{1}{n}A$. Clearly, the (s, t)-entry in B^{-1} is non-zero if and only if it is in $(I - \frac{1}{n}A)^{-1}$. Thus, for maintaining reachability it is sufficient to test whether the (s, t) entry of B^{-1} is non-zero.

More generally we show how to test whether the (i, j)-entry of the inverse B^{-1} of an invertible matrix B equals a number $a \leq n$ using matrix rank. A similar reduction has been used in [14, p. 99]. Let b be the column vector with $b_j = 1$ and all other entries are 0. For every $l \leq n$, the lth entry of the vector $B^{-1}b$ is equal to the (l, j)-entry $(B^{-1})_{l,j}$ of B^{-1}. In particular, the unique solution of the equation $Bx = b$ has $(B^{-1})_{i,j}$ as ith entry. Now let B' be the matrix resulting from B by adding an additional row with 1 in the i-column and otherwise zero. Let further b' be b extended by another entry a. The equation $B'x = b'$ now corresponds to the equations $Bx = b$ and $x_i = a$ and, by the above, this system is feasible if and only if the (i, j)-entry of B^{-1} is equal to a. On the other hand, $B'x = b'$ is feasible if and only if $\text{rank}(B') = \text{rank}(B'|b')$, where $(B'|b')$ is the $(n + 1) \times (n + 1)$ matrix obtained by appending the column b' to B'. As B is invertible, $\text{rank}(B') = \text{rank}(B) = n$ and therefore, we get the following result.

Proposition 3. *Let B be an invertible matrix, $a \leq n$ a number, and B' and b' as just defined. Then, the (i, j)-entry of B^{-1} is equal to a if and only if $\text{rank}(B'|b') = n$.*

Thus, to maintain a small entry of the inverse of a matrix it suffices to maintain the rank of the matrix $B'|b'$ and to test, whether this rank is n (or, otherwise $n+1$). As every change in B yields only one change in $B'|b'$, Algorithm 1 can be easily adapted for this purpose.

By choosing $a = 0$, the following corollary immediately follows from the observation made above, that the (s,t)-entry of the matrix $(nI - A)^{-1}$ is nonzero if and only if there is a directed path from s to t. It implies that also reachability can be maintained.

Corollary 1. *Let G be a directed graph with n vertices, A its adjacency matrix, $B = nI - A$, $a = 0$, and B' and b' as defined above (with s and t instead of i and j). Then, there is a path from node s to node t in G if and only if $rank(B'|b') = n + 1$.*

4 Matrix rank and Reachability in DynFO

In this section we show Theorems 1 and 2. The proofs are based on the algorithms presented in Section 3. The proof for Theorem 3 is given in the full version of the paper. We first give the basic definitions for dynamic complexity, and show that, for domain-independent queries, DynFO programs with empty initialization are as powerful as DynFO programs with $(+,\times)$-initialization. Then we give proof sketches for the two theorems.

4.1 Dynamic Complexity

We basically adopt the original dynamic complexity setting from [16], although our notation is mainly from [20].

In a nutshell, inputs are represented as relational logical structures consisting of a universe, relations over this universe, and possibly some constant elements. For any change sequence, the universe is fixed from the beginning, but the relations in the initial structure are empty. This initially empty structure is then modified by a sequence of insertions and deletions of tuples. The goal of a dynamic program is to answer a given query after each prefix of a change sequence. To this end, the program can use some data structures, represented by auxiliary relations. Depending on the exact setting, these auxiliary relations might be initially empty or might contain some precomputed tuples.

We say that a dynamic program maintains a query q if it has a designated auxiliary relation that always coincides with the query result for the current database. An update program basically consists of two update formulas $\phi^R_{\mathrm{INS}_S}(\vec{x};\vec{y})$ and $\phi^R_{\mathrm{DEL}_S}(\vec{x};\vec{y})$ for each auxiliary relation R, and each input relation S; one for updates of R after insertions of S-tuples and one for deletions of S-tuples. Intuitively, when modifying the tuple $\vec{x}$ with the operation δ, then all tuples $\vec{y}$ satisfying $\phi^R_\delta(\vec{x};\vec{y})$ will be contained in the updated relation R. Besides that, a program might have functions that define the initial values of the auxiliary relations.

Example 1. The transitive closure of an acyclic graph can be maintained by an update program with one binary auxiliary relation T which is intended to store the transitive closure [16,4]. After inserting an edge (u, v) there is a path from x to y if, before the insertion, there has been a path from x to y or there have been paths from x to u and from v to y. Thus, T can be maintained for insertions by the formula $\phi^T_{\mathrm{INS}_E}(u, v; x, y) \stackrel{\text{def}}{=} T(x, y) \vee \big(T(x, u) \wedge T(v, y)\big)$. The formula for deletions is slightly more complicated.

Here, we concentrate on the following three dynamic complexity classes:

- DynFO is the class of all dynamic queries that can be maintained by dynamic programs with formulas from first-order logic starting from an empty database and empty auxiliary relations.
- DynFO(+,×) is defined as DynFO, but the programs have three particular auxiliary relations that are initialized as a linear order and the corresponding addition and multiplication relations. There might be further auxiliary relations, but they are initially empty.
- *Non-uniform* DynFO is defined as DynFO, but the auxiliary relations may be initialized by arbitrary functions.

4.2 DynFO and DynFO(+,×) coincide for domain independent queries

Next, we show that DynFO and DynFO(+,×) coincide for queries that are invariant under insertion and deletion of isolated elements. More precisely, a query q is *domain independent*, if $q(\mathcal{D}_1) = q(\mathcal{D}_2)$ for all databases $\mathcal{D}_1$ and $\mathcal{D}_2$ that coincide in all relations and constants (but possibly differ in the underlying domain). As an example, the Boolean Reachability query is domain independent, as its result is not affected by the presence of isolated nodes (besides s and t).

Theorem 4. *For every domain-independent query q,*
$$q \in \mathsf{DynFO}(+,\times) \text{ if and only if } q \in \mathsf{DynFO}.$$

The proof idea is to simulate several computations. The DynFO program $\mathcal{P}'$ simulates several "runs" of the DynFO(+,×) program $\mathcal{P}$, one for each $m \leq \sqrt{n}$, where n is the domain size. Such a simulation starts as soon as m elements become "active" and serves to answer the query for the period when at least $(m-1)^2$ but less than m^2 elements are active. More details about this construction can be found in the full paper.

Remark 1. Kousha Etessami already observed that arithmetic can be defined incrementally, so that at any point there are relations $<_{ad}$, $+_{ad}$ and $\times_{ad}$ that represent a linear order *on the activated domain*, and the ternary addition and multiplication relations [6].

4.3 Matrix rank in DynFO(+,×)

To the best of our knowledge, computational linear algebra problems like matrix rank and matrix inverse have not been studied before in dynamic complexity

(with the notable exception of Boolean matrix multiplication in [10]). Therefore, there is no standard way of representing the matrix rank problem in the dynamic complexity framework. The key question is how to represent the numbers that appear in a matrix, as their size can grow arbitrarily compared to the dimensions of the matrix. We use a representation that does not allow matrices with large numbers but suffices for our applications in which matrix entries are not larger than the number of rows in the matrix.

More precisely, an input database for the matrix rank query MatrixRank consists of two ternary relations M_+, M_- and a linear order $<$. In the following, we identify the k-th element with respect to $<$ with the number k (and the minimal element represents 1). That the matrix has value $a > 0$ at position (i, j) is represented by a triple (i, j, a) in M_+. Likewise, $a_{ij} = a < 0$ is represented by a triple $(i, j, -a)$ in M_-. For each i, j, at most one triple (i, j, a) can be present in $M_+ \cup M_-$. If, for some i, j there is no triple (i, j, a) then $a_{i,j} = 0$. In this way, we can represent $n \times n$-matrices over $\{-n, \ldots, n\}$ by databases with domain $\{1, \ldots, n\}$. Non-square matrices can be represented as $n \times n$-matrices in a straightforward manner with the help of zero-rows or zero-columns.

Change operations might insert a triple (i, j, a) to M_+ or M_- (in case, no (i, j, b) is there), or delete a triple, but we do not allow change operations on $<$. That is, basically, single matrix entries can be set to 0 or from 0 to some other value. Initially, M_+ and M_- are empty, that is the matrix is the all-zero matrix, but $<$ is a complete linear order. The query MatrixRank maps a database $\mathcal{D}$ representing a matrix A in this way to the set $\{\text{rank}(A)\}$, in case $\text{rank}(A) > 0$ and to $\emptyset$ otherwise.

Theorem 5. MatrixRank *is in* DynFO$(+,\times)$.

There is a subtle technical point in the interpretation of the statement "MatrixRank is in DynFO$(+,\times)$". The database representing the input matrix A comes with a linear order $<_A$ and there is the linear order $<$ initially given to a DynFO$(+,\times)$ program. We require here that these orders are identical (as they are in our applications).

Proof. Algorithm 1 can be extended and translated into a dynamic program for MatrixRank in a straightforward manner. Let DOM be a given domain. In our setting, we have $N = n$, therefore it is sufficient to consider prime numbers $p \leq n^2$. Such prime numbers and arithmetics in $\mathbb{Z}_p$ can be expressed with the help of pairs over DOM, via the bijection $(u_1, u_2) \mapsto (u_1 - 1) \times n + u_2$. □

Remark 2. Due to the initial linear order, MatrixRank does not fit into the domain independence framework of Theorem 4. To maintain matrix rank in DynFO, we would need to build $<$ incrementally when entries are inserted to the matrix. However, when we use MatrixRank to maintain a domain independent query, Theorem 4 yields a DynFO upper bound.

4.4 Reachability in DynFO

As described at the beginning of this section, the reachability query Reach has a straightforward, and standard formalization in the dynamic complexity frame-

work. Now we can sketch the proof of the main result of this paper.

Theorem 1 ***(restated).*** REACH *is in* DynFO.

Proof. It is straightforward to transform the approach of Proposition 3 into a dynamic program with arithmetic. The edge relation E can be viewed as an adjacency matrix A for the graph, and $nI - A$ and then $B'|b'$ can be easily defined in first-order logic. We note that each change of one pair in E only changes one entry in $B'|b'$. It thus suffices to maintain $\text{rank}(B'|b')$ by the program of Theorem 5 to maintain reachability from s to t. This shows REACH $\in$ DynFO$(+,\times)$. As the reachability query is domain independent, Theorem 4 yields the theorem. □

Remark 3. In the DynFO-framework considered here, elements cannot be removed from the domain. Removal of nodes is allowed in the FOIES-framework of Dong, Su and Topor: when a node is not used in any edge, then it is removed from the domain. The proof above can be adapted to this framework.

By simple reductions we obtain the following further results.

Theorem 6. *(a) Regular path queries in directed labeled graphs can be maintained in* DynFO*. (b) Conjunctions of regular path queries in directed labeled graphs can be maintained in* DynFO*. (c)* 2*-SAT is in* DynFO*.*

5 Conclusion

The main technical contribution of the paper is that maintaining the rank of a matrix is in DynFO$(+,\times)$. From this we derive that Reachability can be maintained in DynFO improving on both the complexity and uniformity of previous results [9,3]. In the case of matching, we are able to prove only a non-uniform bound. As exemplified by regular path queries and 2-SAT, the fact that Reachability is in DynFO may help to show that many other queries and problems can be maintained in DynFO. However, the DynFO bound obtained here does not extend to all of NL, simply because DynFO is not known to be closed under even *unbounded* first order projection reductions. We believe that also the approach through Linear Algebra might yield further insights.

It is an interesting open question whether a Reachability witness can be maintained in DynFO and whether a Shortest Path witness can be maintained in DynTC^0. Some further reflections on our results can be found in the full version of the paper.

6 Acknowledgements

We would like to thank William Hesse for stimulating and illuminating discussions. We are grateful to the anonymous referee who brought [7] to our attention. Further we thank Nils Vortmeier for proofreading. The first and the third authors were partially funded by a grant from Infosys Foundation. The second author

is supported by the Singapore National Research Foundation under NRF RF Award No. NRF-NRFF2013-13. The last two authors acknowledge the financial support by DFG grant SCHW 678/6-1.

References

1. Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.
2. Michael Artin. *Algebra*. Featured Titles for Abstract Algebra. Pearson, 2010.
3. Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *ICALP (1)*, pages 356–367, 2014.
4. Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*, 120(1):101–106, July 1995.
5. 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.
6. Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS*, pages 235–243, 1998.
7. Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. Dynamic matrix rank. *Theor. Comput. Sci.*, 410(41):4085–4093, 2009.
8. Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT*, pages 236–248, 2012.
9. William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003.
10. William Hesse and Neil Immerman. Complete problems for dynamic complexity classes. In *LICS*, page 313, 2002.
11. Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010.
12. Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.
13. Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
14. Bastian Laubner. *The structure of graphs and new logics for the characterization of Polynomial Time*. PhD thesis, Humboldt University of Berlin, 2011.
15. Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
16. Sushant Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, October 1997.
17. John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22(2):347–371, 1997.
18. Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *FOCS*, pages 509–517, 2004.
19. Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126, 2007.
20. Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015.