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 [27]. The proofs rely mainly on elementary Linear Algebra. The second result can also be concluded from [13].

1 Introduction

Dynamic Complexity Theory studies dynamic problems from the point of view of Descriptive Complexity (see [21]). 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 [27].

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 [5,10,15,17,27,33]. In this paper, we confirm the conjecture of Patnaik and Immerman [27] 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 [17], 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 [5] 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 [29,31,**?**]. The usefulness of matrix rank for graph problems in a logical framework has been demonstrated in [23]. Both [31,23] 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 [13]. 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 [31,**?**] 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 [26] 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) [9]. 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, reachability and maximum matching 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). In Subsection 3.1, we describe how to maintain the rank of a matrix. In Subsection 3.2 we describe how to maintain an entry of the inverse of a matrix by a reduction to the rank of a matrix and we show that this immediately yields an algorithm for Reachability in directed graphs. In Subsection 3.3 we give non-uniform dynamic algorithms for Existence of perfect matching and Size of maximum matching, respectively.

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. For simplicity, we describe the algorithm for integer matrices although 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 $(A \bmod p)$ 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 $(A' \bmod p)$ 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 $(A \bmod p)$ is at least k. Thus in order to compute the rank of A it suffices to compute the rank of $(A \bmod p)$ in parallel for the primes in $[\max(n, \log N)^3]$, and to take the maximum over all such ranks.

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

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$(A) = n - |B \cap K|$.*

Proof. It is well-known that $\mathrm{rank}(A) = n - \dim(K)$. To prove the proposition it therefore suffices to show that $B \cap K$ is a basis for K. To this end, let u be an arbitrary vector from K and $u = \sum_{v \in B} b_v v$. Let us assume towards a contradiction that $b_v \neq 0$, for some $v \in B - K$. Let $i = \mathrm{pc}(v)$. By definition the i-th coordinate of Av and therefore also of $Ab_v v$ is non-zero. However, as $(Aw)_i = 0$, for all other $w \in B$, we can conclude that $(Au)_i \neq 0$, the desired contradiction. Therefore, $u \in \mathrm{span}(B \cap K)$ and therefore $B \cap K$ is a basis for 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

- $U = \{u\}$ 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 $\mathbb{Z}_p^m$.*

Proof. We first note that if we have two vectors $v \neq w$ in some basis of $\mathbb{Z}_p^m$ and replace w by $x \stackrel{\text{def}}{=} w - (A'w)_i (A'v)_i^{-1} v$ then we get again a basis and $S'(x) \subseteq (S'(v) \cup S'(w)) - \{i\}$. The former ensures B' is again a basis of $\mathbb{Z}_p^m$ after the construction above.

It thus only remains to show that B' is A'-good. For this we first observe that $i \notin S'(v)$ for all vectors v added to B' in Steps (1b) and (2b). For Step (2b) this is the case because $i \notin S'(\hat{u})$ and after (1b) also $i \notin S'(v)$. Furthermore $k \notin S'(v)$ for all vectors v added to B' in Step (2b).

We now show by a case distinction that all elements x of $B' - K'$ are j-unique for some j. We observe that x can not be one of the vectors added in Step (1bii) since those are are actually in K'. For the same reason x cannot be the vector $\hat{u}$ added in Step (1biii) if $S'(u) = \{i\}$.

The remaining cases are as follows:

- If $x = \hat{v}$ then x is i-unique by Steps (1bi)-(1biii)

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 $u \in B$)

- If $x = \hat{u}$ and $S'(\hat{u}) \neq \emptyset$ then x is k-unique by Step (2b).
- If x is an element added in Step (1bi) (of the form $w-(A'w)_i(A'\hat{v})_i^{-1}\hat{v}$), then w is ℓ-unique with respect to B and A, for some $\ell \neq i$ (or $\ell \notin \{i,k\}$ if k is defined) as $i \in S(u)$ (or $\{i,k\} \subseteq S(u)$, respectively). However, then $\ell \in S'(x)$ and no other vector y with $\ell \in S'(y)$ can be in B'. Thus x is ℓ-unique.
- If x is any other element then it has already been in $B - K$ and thus it is ℓ-unique with respect to B and A, for some $\ell \neq i$ (or $\ell \notin \{i,k\}$ if k is defined). As before, no other vector y with $\ell \in S'(y)$ can be in B'.

□

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 [13] in a context where parallel complexity was not considered. Indeed, both algorithms essentially maintain Gaussian elimination, but the algorithm in [13] maintains a stronger normal form (reduced row echelon form) that differs by multiplication by a permutation matrix from our form. However, Algorithm 1 in [13], 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. [20, Theorem 6.1.10.]) and its inverse can be expressed by its Neumann series as follows.

$$(I - \frac{1}{n}A)^{-1} = I + \sum_{i=1}^{\infty} (\frac{1}{n}A)^i.$$

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 [23, 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

$$\begin{aligned} Bx &= b \\ x_i &= a \end{aligned}$$

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 $\operatorname{rank}(B') = \operatorname{rank}(B'|b')$, where $(B'|b')$ is the $(n + 1) \times (n + 1)$ matrix obtained by appending the column b' to B'. As B is invertible, $\operatorname{rank}(B') = \operatorname{rank}(B) = n$ 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 $\operatorname{rank}(B'|b') = n$.*

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 non-zero 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 $\mathit{rank}(B'|b') = n+1$.*

3.3 Maintaining Matching

We first show how to non-uniformly maintain whether a graph has a perfect matching, afterwards we extend the technique for the maintenance of the size of a maximum matching.

The basic idea for maintaining whether a graph has a perfect matching relies on a correspondence between the determinant of the Tutte matrix of a graph and the existence of perfect matchings. The *Tutte matrix* T_G of an undirected graph G is the $n \times n$ matrix with entries

$$t_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{ji} & \text{if } (i,j) \in E \text{ and } i > j \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

where the x_{ij} are indeterminates.

Theorem 4 (Tutte [32]). *A graph G has a perfect matching if and only if* $\det(T_G) \neq 0$.

We note that $\det(T_G)$ is a polynomial with variables from $\{x_{i,j} \mid 1 \leq i \leq j \leq n\}$ with possibly exponentially many terms. However, as we will see, whether $\det(T_G)$ is the zero-polynomial can be tested by evaluating the polynomial for well-chosen positive integer values. For a graph G, let w be a function that assigns a positive integer weight to every edge (i,j) and let $B_{G,w}$ be the integer matrix obtained from T_G by substituting x_{ij} by $2^{w(i,j)}$. By Tutte's Theorem, if G has no perfect matching then $\det(B_{G,w}) = 0$.

Theorem 5 (Mulmuley, Vazirani and Vazirani [26]). *Let G be a graph with a perfect matching and w a weight assignment such that G has a unique perfect matching with minimal weight with respect to w. Then* $\det(B_{G,w}) \neq 0$.

Using the technique implicit in [30] one can find, for every $n \in \mathbb{N}$, weighting functions $w^1, \ldots, w^{n^2}$ with weights in $[4n]$, such that for every graph G there is an $i \in [n^2]$ such that if G has a perfect matching, then it has a unique minimal weight matching with respect to w^i.

For the sake of completeness, we show how to obtain those functions. The following lemma is due to Mulmuley, Vazirani and Vazirani [26], but we use the version stated in [22].

Lemma 1 (Isolation Lemma). *Given a non-empty $\mathcal{F} \subseteq 2^{[n]}$. If a weight assignment $w \in [N]^{[n]}$ is uniformly chosen at random, then with probability at least $1 - \frac{n}{N}$, the minimum weight subset in $\mathcal{F}$ is unique; where the weight of a subset $F \in \mathcal{F}$ is $\sum_{i \in F} w(i)$.*

Lemma 2 (Non-uniform Isolation Lemma, implicit in [30]). *Let $m \in \mathbb{N}$ and $\mathcal{F}_1, \ldots, \mathcal{F}_{2^m} \subseteq 2^{[m]}$. There is a sequence $w^1, \ldots, w^m$ of weight assignments from $[4m]^{[m]}$ such that for any $i \in [2^m]$ there exists a $j \in [m]$ such that the minimum weight subset of $\mathcal{F}_i$ with respect to w^j is unique.*

Proof. The proof is implicit in the proof of Lemma 2.1 in [30]. For the sake of completeness we give a full proof.

We call a sequence of weight assignments $u^1, \ldots, u^m$ bad for some $\mathcal{F}_i$ if no $F \in \mathcal{F}_i$ is a minimum weight subset with respect to any u^j. For each $\mathcal{F}_i$ the probability of a randomly chosen weight sequence $U \stackrel{\text{def}}{=} u^1, \ldots, u^m$ to be bad is at most $(\frac{1}{4})^m$ thanks to Lemma 1 (for $N \stackrel{\text{def}}{=} 4m$). Thus the probability that such a U is bad for *some* $\mathcal{F}_i$ is at most $2^m \times (\frac{1}{4})^m < 1$. Hence there exists a sequence U which is good for all $\mathcal{F}_i$. □

We immediately get the following corollary.

Corollary 2. *Let $G_1, \ldots, G_{2^{n^2}}$ be some enumeration[5] of the graphs on $[n]$ and let $\mathcal{F}_1, \ldots, \mathcal{F}_{2^{n^2}}$ be their respective sets of perfect matchings. There is a sequence $w^1, \ldots, w^{n^2}$ of weight assignments to the edges of $[n]^2$ such that for every graph G over $[n]$ there is some $i \in [n^2]$ such that if G has a perfect matching then it also has a perfect matching with unique minimal weight with respect to w^i.*

Then, in order to decide whether a graph G over $[n]$ has a perfect matching, it is sufficient to maintain whether $\det(B_{G,w^i}) \neq 0$ for any $i \in [n^2]$. As the determinant $\det(B_{G,w^i})$ is at most $n!(2^{4n})^n$, it thus suffices, for every i, to maintain whether $\det(B_{G,w^i}) \neq 0$ modulo p, and thus to maintain the rank of B_{G,w^i} modulo p, for polynomially many primes p. As the rank of a matrix can be maintained dynamically as shown in Subsection 3.1 (and as each change in the graph yields only one change in each matrix) we altogether get a non-uniform procedure for dynamically testing whether a graph has a perfect matching. The non-uniformity is due to the choice of the weight assignments. We note that, although the entries in B_{G,w^i} might be of exponential size in n, the dynamic algorithm only needs to maintain matrices with numbers modulo small primes.

The algorithm for the more general problem of maintaining the size of a maximum matching relies on an extension of Tutte's theorem by Lovász [24]. We state the version of this theorem from [28].

[5] For notational simplicity we use n^2 instead of $\binom{n}{2}$, here.

Theorem 6 (Lovász). *Let G be a graph with a maximum matching of size m. Then rank$(T_G) = 2m$.*

Theorem 7. *Let G be a graph with a maximum matching of size m, and let w be a weight assignment for the edges of G such that G has a maximum matching with unique minimal weight with respect to w. Then rank$(B_{G,w}) = 2m$.*

This theorem is implicit in Lemma 4.1 in [19]. For the sake of completeness we give a full proof here.

Proof (of Theorem 7). Recall that the rank of a matrix can be defined as the size of the largest submatrix with non-zero determinant. Thus $\text{rank}(B_{G,w}) \leq \text{rank}(T_G)$, and therefore $\text{rank}(B_{G,w}) \leq 2m$ by Theorem 6.

For showing $\text{rank}(B_{G,w}) \geq 2m$ we adapt the proof of Theorem 6 given in [28]. Let U be the set of vertices contained in the maximum matching of G with minimal weight, and G' the subgraph of G induced by U. Observe that G' has a perfect matching and that its weight with respect to w is unique. Restricting $B_{G,w}$ to rows and columns labeled by elements from U yields the matrix $B_{G',w'}$ where w' is the weighting w restricted to edges from G'. However, then $\det B_{G',w'} \neq 0$ by Theorem 5 and therefore $\text{rank}(B_{G,w}) \geq 2m$. □

An easy adaption of Corollary 2 to maximum matchings and the same construction as above yields a procedure for maintaining the size of a maximum matching.

4 Matrix rank and Reachability in DynFO

In this section we show Theorems 1, 2 and 3. The proofs are based on the algorithms presented in Section 3.

We first give the basic definitions for dynamic descriptive complexity and, in particular, DynFO in Subsection 4.1. In Subsection 4.2 we show that, for domain-independent queries, DynFO programs with empty initialization are as powerful as DynFO programs with $(+, \times)$-initialization. Then we show how to maintain the rank of a matrix in DynFO$(+,\times)$ (in Subsection 4.3), the Reachability query, regular path queries and 2-SAT in DynFO (in Subsection 4.4), and maximum matching in non-uniform DynFO (in Subsection 4.5).

4.1 Dynamic Complexity

We basically adopt the original dynamic complexity setting from [27], although our notation is mainly from [33].

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. As much of the

original motivation for the investigation of dynamic complexity came from incremental view maintenance (cf. [11,8,27]), it is common to consider such a logical structure as a relational database and to use notation from relational databases. As an example, for the reachability problem, the database $\mathcal{D}$ has domain DOM containing the nodes of the graph. It has one relation E, representing the edges, and two constants, s and t. The reachability problem itself is then represented by the *Boolean query* REACH whose result is TRUE if there is a path from s to t in the graph (DOM, E), and FALSE, otherwise. 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. We now give more precise definitions.

A *dynamic instance* of a query q is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a finite database over a finite domain DOM and α is a sequence of updates to $\mathcal{D}$, i.e. a sequence of insertions and deletions of tuples over DOM. The dynamic query $\text{DYN}(q)$ yields as result the relation that is obtained by first applying the updates from α to $\mathcal{D}$ and then evaluating q on the resulting database.

The database resulting from applying an update δ to a database $\mathcal{D}$ is denoted by $\delta(\mathcal{D})$. The result $\alpha(\mathcal{D})$ of applying a sequence of updates $\alpha = \delta_1 \dots \delta_\ell$ to a database $\mathcal{D}$ is defined by $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \delta_\ell(\dots(\delta_1(\mathcal{D}))\dots)$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (input) database $\mathcal{D}$, an initial state with initial auxiliary data. The latter defines the new state of the dynamic program for each possible update δ.

A *dynamic schema* is a tuple $(\tau_{\text{in}}, \tau_{\text{aux}})$ where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. In this paper the auxiliary schemas are purely relational, while the input schemas may contain constants.

Definition 1. *(Update program) An* update program $\mathcal{P}$ *over dynamic schema* (τ_{in}, τ_{aux}) *is a set of first-order formulas (called update formulas in the following) that contains, for every* $R \in \tau_{aux}$ *and every* $\delta \in \{\text{INS}_S, \text{DEL}_S\}$ *with* $S \in \tau_{in}$*, an update formula* $\phi^R_\delta(\vec{x}; \vec{y})$ *over* $\tau_{in} \cup \tau_{aux}$ *where* $\vec{x}$ *and* $\vec{y}$ *have the same arity as* S *and* R*, respectively.*

The semantics of update programs is defined below. 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.

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 [27,9]. 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.

The semantics of update programs is made precise now. A *program state* $\mathcal{S}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(\mathcal{D}, \mathcal{A})$ where $\mathcal{D}$ is a database over the input schema (the *current database*) and $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*), both with domain DOM. The effect $P_\delta(\mathcal{S})$ of an update $\delta(\vec{a})$, where $\vec{a}$ is a tuple over DOM, to a program state $\mathcal{S} = (\mathcal{D}, \mathcal{A})$ is the state $(\delta(\mathcal{D}), \mathcal{A}')$, where $\mathcal{A}'$ consists of relations $R' \stackrel{\text{def}}{=} \{\vec{b} \mid \mathcal{S} \models \phi^R_\delta(\vec{a}; \vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of an update sequence $\alpha = \delta_1 \dots \delta_l$ to a state $\mathcal{S}$ is the state $P_{\delta_l}(\dots(\mathcal{P}_{\delta_1}(\mathcal{S}))\dots)$.

Definition 2. *(Dynamic program) A* dynamic program *is a triple* $(P, \textsc{Init}, Q)$*, where*

- P *is an update program over some dynamic schema* (τ_{in}, τ_{aux})*,*
- INIT *is a mapping that maps* τ_{in}*-databases to (initial)* τ_{aux}*-databases, and*
- $Q \in \tau_{aux}$ *is a designated* query symbol*.*

A dynamic program $\mathcal{P} = (P, \textsc{Init}, Q)$ *maintains* a dynamic query $\textsc{Dyn}(q)$ if, for every dynamic instance $(\mathcal{D}, \alpha)$, where in $\mathcal{D}$ all relations are empty, the relation $q(\alpha(\mathcal{D}))$ coincides with the query relation $Q^{\mathcal{S}}$ in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\textsc{Init}}(\mathcal{D}))$, where $\mathcal{S}_{\textsc{Init}}(\mathcal{D})$ is the initial state, i.e. $\mathcal{S}_{\textsc{Init}}(\mathcal{D}) \stackrel{\text{def}}{=} (\mathcal{D}, \textsc{Init}_{\text{aux}}(\mathcal{D}))$.

Several dynamic settings and restrictions of dynamic programs have been studied in the literature (see e.g. [27,12,16,14]). Here, we concentrate on the following three classes, whose relationship with circuit complexity classes has already been mentioned in the introduction.

- 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$(+,\times)$ 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$(+,\times)$ coincide for domain independent queries

Next, we show that DynFO and DynFO$(+,\times)$ 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 8. *For every domain-independent query q the following are equivalent:*

(1) $q \in \mathsf{DynFO}(+,\times)$*;*
(2) $q \in \mathsf{DynFO}$*.*

Proof. Of course, we only need to prove that (1) implies (2). We give the proof only for Boolean graph queries. The generalization to databases with arbitrary signature and non-Boolean queries is straightforward.

Let q be a domain-independent query and $\mathcal{P}$ a $\mathsf{DynFO}(+,\times)$ program that maintains q. We recall that change sequences are applied to an initially empty graph but that $\mathcal{P}$ has a linear order and the corresponding addition and multiplication relations available.

Let in the following n denote the size of the domain DOM. As there is a linear order $<$ on DOM we can assume that DOM is of the form $[n]$ and that $<$ is just the usual linear order on $[n]$. Likewise, if there is a linear order available on a subset of DOM with j elements, we can assume for simplicity that this set is just $[j]$.

We say that an element u of the universe has been *activated* by a change sequence $\alpha = \delta_1, \ldots, \delta_\ell$, if u occurs in some δ_i, no matter, whether an edge with a is still present in $\alpha(\mathcal{D})$. We denote the set of activated elements by A.

We will construct a DynFO program $\mathcal{P}'$ that simulates $\mathcal{P}$. By definition of DynFO, $\mathcal{P}'$ has to maintain q under change sequences from an initially empty graph (just as $\mathcal{P}$), but with initially empty auxiliary relations (unlike $\mathcal{P}$).

It is well known that arithmetic on the active domain can be constructed on the fly when new elements are activated [12]. Yet this is not sufficient for simulating $\mathcal{P}$: thanks to its built-in arithmetic, $\mathcal{P}$ can maintain complex auxiliary structures even for elements that have not been activated so far. On the other hand, the program $\mathcal{P}'$ can only use elements as soon as they are activated[6]. Thus, the challenge for the construction of $\mathcal{P}'$ is to make arithmetic as well as the auxiliary data for an element available as soon as it is activated.

The basic idea for the construction of $\mathcal{P}'$ is to start simulating $\mathcal{P}$ for active domains of size m^2 as soon as m elements are activated. There will be one such simulation for every m with $(m-1)^2 < n^2$, in parallel. For each m, the "m-simulation" starts from an initially empty database and simulates $\mathcal{P}$ for an insertion sequence leading to the current database. The goal is that as soon as $(m-1)^2$ elements are activated, the m-simulation will be "consistent" with $\mathcal{P}$. We now describe this basic idea in more detail.

Let $\mathcal{D}$ be the initial empty graph on $[n]$ and $\alpha = \delta_1, \ldots, \delta_\ell$ a change sequence. The m-simulation (i.e., the simulation for domain size m^2) begins, as soon as $\geq m$ elements have been activated. For simplicity we assume that $[m]$ is the set of these elements. Let $\mathcal{D}_m = \alpha'(\mathcal{D})$, where α' is the shortest prefix of α such that $\alpha'(\mathcal{D})$ has at least m activated elements.

[6] Non-activated elements are already present in the domain before they are activated, yet it is easy to see that all non-activated elements behave similarly since they are all updated by the same first-order formulas. Therefore they cannot be used for storing complex auxiliary data structures.

In the m-simulation of $\mathcal{P}$ elements of $[m^2]$ are encoded by pairs over $[m]$. The simulation uses an auxiliary edge relation E'_m over $[m]^2$, which is initially empty, the linear order on $[m]$ and the corresponding addition and multiplication relations, and all other auxiliary relations of $\mathcal{P}$, all of them initially empty. The arity of all these relations used by $\mathcal{P}'$ is twice the one in $\mathcal{P}$ due to the encoding of $[m^2]$ by pairs.

For each of the subsequent change operations δ (as long as necessary), the m-simulation inserts four edges from $\mathcal{D}_m$ to E'_m and applies δ to E'_m. If δ deletes an edge that has not yet been transferred from $\mathcal{D}_m$ to E'_m then additionally this edge is deleted in $\mathcal{D}_m$. For all these (up to) five change operations, $\mathcal{P}'$ also applies the corresponding updates to the auxiliary relations and deletes the four inserted edges from $\mathcal{D}_m$.

As soon as more than $(m-1)^2$ elements are activated, we can be sure that all edges of $\mathcal{D}_m$ have been inserted to E'_m. Thus, the m-simulation becomes the "main simulation" — until the $(m+1)$-simulation takes over, when more than m^2 elements are activated. During that time, the query relation Q' of $\mathcal{P}'$ always has the same value as the relation Q'_m corresponding to the designated query relation of $\mathcal{P}$. The correspondence between the simulation on $[m]^2$ and the actual domain is induced by the bijection $(u_1, u_2) \mapsto (u_1 - 1) \times m + u_2$.

So far, $\mathcal{P}'$ would need, for every $m \leq n$ a separate collection of auxiliary relations, which is of course, not possible for a dynamic program. However, all these relations can be combined into one (of each kind), by increasing the arity and prefixing each tuple by the defining element m. As an example, all relations E'_m are encoded into one 5-ary relation E' and E'_m is just the set of pairs $((u_1, u_2), (v_1, v_2))$, for which (m, u_1, u_2, v_1, v_2) is in E'.

We now describe $\mathcal{P}'$ in more detail. We describe first, how $\mathcal{P}'$ constructs a linear order[7] $<$, an addition relation $+$ and a multiplication relation $\times$ on the set A of activated elements. We note that this part of the simulation is just as in [12]. We assume without loss of generality that $\mathcal{P}$ never changes its linear order, addition relation and multiplication[8].

The relation $<$ will be, at any time, a linear order of all currently activated elements, in the order of activation. For concreteness: if an edge (a, b) is inserted which activates a and b then $a < b$ become the two largest elements of $<$.

We always identify activated elements with their position in $<$, that is, the minimal element in $<$ is considered as 0, the second as 1 and so on. We use numbers as constants in formulas. It is straightforward to replace these numbers by "pure" formulas. For example, the subformula $x > 1$ can be replaced by $\exists x_1 \exists x_2\ x_1 < x_2 \wedge x_2 < x$.

The update formulas for these three relations are straightforward. For delete operations, nothing has to be changed, that is, e.g., $\phi^{<}_{\text{DEL}}(a, b; x, y) = x < y$ and likewise for the other two relations.

The update formula $\phi^{<}_{\text{INS}}(a, b; x, y)$ for insertions states that

[7] We use infix notation for $<$, $+$ and $\times$.

[8] Otherwise, they have to be duplicated.

- $x < y$; or
- x is already activated, $y \in \{a, b\}$ is not yet activated; or
- $x = a$, $y = b$, $a \neq b$, and both a and b are not yet activated.

The formulas for $+$ and $\times$ are in the same spirit and use the well-known inductive definitions of addition and multiplication, respectively. We note that $<$, $+$, $\times$ can be lifted to relations over pairs in a straightforward fashion. Here, a pair (u_1, u_2) over $[m]^2$ corresponds to the number $(u_1 - 1) \times m + u_2$.

Algorithm 2 describes the update operations that are induced by a change operation δ in the simulation over $[m]^2$ in an algorithmic fashion. As the algorithm only performs a constant number of steps for each change operation, it is straightforward (though tedious) to turn it into a bunch of first-order update formulas.

If δ is an insertion of an edge $e = (u, v)$ into the graph, Algorithm 2 inserts the corresponding edge $\hat{e}$ into the graph over $[m]^2$. Here, $\hat{e}$ is defined as follows. Let u_1, u_2, v_1, v_2 be the uniquely determined elements from $[m]$ such that $u = (u_1 - 1) \times m + u_2$ and $v = (v_1 - 1) \times m + v_2$. Then, $\hat{e} = ((u_1, u_2), (v_1, v_2))$.

The correctness of $\mathcal{P}'$ can be established as follows. Let $\alpha = \delta_1, \ldots, \delta_\ell$ be a change sequence with universe $[n]$. Let p be the number of activated elements after applying α to the empty graph. Let m be chosen such that $(m-1)^2 < p \leq m^2$. At the time of the activation of the $(m+1)$st element, $\mathcal{D}_m$ has at most $[m]^2$ edges. These edges can thus be added to E'_m within the subsequent $\frac{m^2}{4}$ steps. On the other hand, there are at least $\frac{(m-1)^2-(m+1)}{2}$ insertion steps from α needed to reach the point of more than $(m-1)^2$ activated elements. It is straightforward to check that $\frac{(m-1)^2-(m+1)}{2} \geq \frac{m^2}{4}$ for $m \geq 8$ and therefore the edge transfer is completed before the m-simulation becomes the "main simulation". For each fixed universe size ≤ 64, q can be maintained by separate formulas that can then be combined with the formulas derived from Algorithm 2.

As the m-simulation starts at a moment when arithmetic is available for $[m]^2$, it can be shown by a straightforward induction that $\mathcal{P}'$ indeed simulates $\mathcal{P}$ on a universe of size m^2 and thus delivers the same answer as $\mathcal{P}$ would do. As during the time in which the m-simulation is the "main simulation" the graph has at most m^2 vertices, the result returned by $\mathcal{P}$ is the same as it would be on universe $[n]$, thanks to the domain independence of q. □

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 [12].

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 [18]). Therefore,

Algorithm 2 Updates for change operation δ relative to universe size m^2

if $|A| \leq m$ after change δ **then** ▷ Wait until universe size $> m$
  $P_m := E$ (after application of δ)
  $E'_m := \emptyset$
  for all $R \in \tau_{\text{aux}}$ **do**
    $R'_m := \emptyset$
  end for
else ▷ Simulate $\mathcal{P}$ over universe $[m]^2$
  for $i \in \{1, 2, 3, 4\}$ **do** ▷ Transfer four edges from G_m to E'_m
    $e :=$ "smallest" edge from P_m
    Delete e from P_m
    Add $\hat{e}$ to E'_m
    for all $R \in \tau_{\text{aux}}$ **do**
      Apply the update formula for insertion of $\hat{e}$ to R'_m
    end for
  end for
  ▷ Apply the current change to E'_m
  if the current change operation inserts an edge e **then**
    Add $\hat{e}$ to E'_m
    for all $R \in \tau_{\text{aux}}$ **do**
      Apply the update formula for insertion of $\hat{e}$ to R'_m
    end for
  end if
  if the current change operation deletes an edge e **then**
    Delete $\hat{e}$ from E'_m
    for all $R \in \tau_{\text{aux}}$ **do**
      Apply the update formula for deletion of $\hat{e}$ to R'_m
    end for
    if $e \in [m]^2$ **then** Remove e from P_m
  end if
  if $(m-1)^2 < |A| \leq m^2$ **then** $Q' := Q'_m$
end if

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 2 ***(restated).*** MatrixRank *is in* $\mathsf{DynFO}(+,\times)$.

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

Proof. We describe how Algorithm 1 can be extended and translated into a dynamic program for MatrixRank. 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 $(u_1, u_2) \mapsto (u_1 - 1) \times n + u_2$. In our notation, we use single variables for numbers $\leq n^2$, thus representing two variables over DOM. The dynamic program for MatrixRank uses four auxiliary relations (beyond $<$, $+$ and $\times$) with the following intention.

- Relation B shall contain a tuple[9] (p, i, j, a) if p is a prime number and, in the run of the dynamic algorithm modulo p, the i-th vector $b(p, i)$ of the basis has the j-entry a. (Again, $b(p, i)_j = 0$ is encoded by the absence of tuples $(p, i, j, \cdot)$);
- Relation C shall contain a tuple (p, i, j, a) if p is a prime number and, in the run of the dynamic algorithm modulo p, the j-th entry of the vector $A \times b(p, i)$ is a modulo p, where A denotes the current matrix;
- Relation R shall contain (p, k) if $n - k$ is the number of basis vectors in the kernel K of A modulo p.
- Relation Q shall contain the maximal element k such that (p, k) is in R for some p.

Whether a number p is a prime number can be tested by a first-order formula thanks to the availability of $\times$. Initially[10], for each prime p, all vectors of the form $(p, i, i, 1)$ are put into B. As the matrix A is initially all-zero, C is initially empty. Relation R and Q are initially empty, reflecting that $\text{rank}(A) = 0$ for all primes p.

We note that we do not need to represent, for basis vectors v, $S(v)$ and $\text{pc}(v)$ as both can be inferred from B and C in a first-order fashion. That a basis is

[9] We note that these tuples are 6-ary, as p and a represent pairs.

[10] More precisely: the first step of the computation (which sees an empty B) takes into account that B consists of these tuples.

A-good can also be tested, but we do not need such a test, as this property is guaranteed by the algorithm. Furthermore the relation Q can be inferred from R straightforwardly. It remains to show how, for any p, the computation of B' from B, as described in Algorithm 1, can be done by a dynamic program.

We adopt the notation from Algorithm 1. First of all, C can be easily updated as in one step only one entry of A changes. The sets U, V, W can be easily represented by first-order formulas. By inspection of Algorithm 1, it is easy to see that each of the steps can be expressed by a first-order formula. By composition of these formulas we can obtain first-order update formulas for B. It is important to observe, that each vector from B yields one vector of B'. Therefore, the numbering of vectors in B (2nd component of (p, i, j, a)) can be maintained. Furthermore, at most one vector from V moves from K to $B' - K'$ and only u might move from $B - K$ to K'. All other vectors stay in their part of the basis. Therefore, R can be easily updated as well. This completes the description of the dynamic program for MatrixRank. Its correctness follows from the correctness of the underlying dynamic algorithm, that is, from Propositions 1 and 2. □

Remark 2. Due to the initial linear order, MatrixRank does not fit into the domain independence framework of Theorem 8. 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 8 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 framework. 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 2to maintain reachability from s to t. This shows Reach $\in$ DynFO$(+,\times)$. As the reachability query is domain independent, Theorem 8 yields the theorem. □

Remark 3. By a straightforward modification, the binary reachability query can also be maintained. This query returns all pairs (u, v) of a graph, for which there is a path from u to v. The above construction is basically done for each pair (u, v) in the role of (s, t), at the cost of an arity increase by two for the non-arithmetic auxiliary relations.

Remark 4. 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 9. *(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.

Proof. Towards (a) and (b), a labeled directed graph $G = (V, E, \Sigma)$ is represented by a set V of nodes, an alphabet Σ of possible labels, and a ternary relation E consisting of all triples (v, a, v') for which there is an a-labeled edge from v to v'.

For the maintainability in DynFO, the regular expression α is fixed and edges in the graph may be added, deleted or relabeled. The question is whether, for two nodes s and t, t is reachable from s via a (not necessarily simple) path, whose label sequence yields a word in the regular language $L(\alpha)$. That such regular path queries can be maintained in DynFO can be shown by the following simple reduction to Reachability.

For a given regular expression α, we fix an NFA $M = (Q, \Sigma, \delta, q_0, f)$ for $L(\alpha)$ with a unique accepting state f. To maintain the query α in the labeled graph $G = (V, E, \Sigma)$, the algorithm maintains Reachability from (s, q_0) to (t, f) in the unlabeled graph $G \times M$ with node set $V \times Q$ and edge set $\{\big((v, q), (v', q')\big) \mid (v, a, v') \in E, (q, a, q') \in \delta\}$. This reduction from RPQ α to Reachability can be easily defined in first-order logic and each change in G induces at most $|Q|$ (and thus a bounded number of) changes in $G \times M$. Thus we have the following result. Statement (b) follows as DynFO is closed under Boolean operations.

For (c), instances of 2-satisfiability are represented as structures as follows. The domain of a structure representing a formula φ is the set of variables of φ. The clauses of φ are represented by three binary input relations C_{TT}, C_{TF} and C_{FF} such that a tuple $(x, y) \in C_{TT}$ corresponds to a clause $x \vee y$, a tuple $(x, y) \in C_{TF}$ to a clause $x \vee \neg y$, and a tuple $(x, y) \in C_{FF}$ to a clause $\neg x \vee \neg y$.

The 2-satisfiability problem can be easily maintained in DynFO using the maintainability of reachability and the standard reduction from 2-satisfiability to reachability. This reduction maps a formula φ with variables V to the graph $G = (V \cup \overline{V}, E)$ where $\overline{V} = \{\neg x \mid x \in V\}$ and E contains the edges $\neg L \to L'$ and $\neg L' \to L$ if $L \vee L'$ is a clause in φ. It can be easily seen that φ is satisfiable if and only if there is no variable $x \in V$ such that there are both a path from x to $\neg x$ and a path from $\neg x$ to x in G. Furthermore the modification of a single clause in φ induces only two first-order definable modifications to the edge set of the corresponding graph. □

Remark 5. The (informally presented) reduction from Reachability to matrix rank (as well as the other reductions in this paper) can be formally defined

as bounded-expansion first-order reductions [27]. In a nutshell, a problem q is bounded-expansion first-order reducible to another problem q' if a single modification in instances of q induces a bounded number of modifications in instances of q'. If q is bounded-expansion first-order reducible to q' and q' is in DynFO then $q \in$ DynFO, as well.

4.5 Matching in non-uniform DynFO

The matching problem, like the MatrixRank problem, has not been studied in dynamic descriptive complexity before. The query MaxMatching maps a database with a single binary relation that represents a graph G to the set $\{k\}$ where k is the size of a maximum matching of G.

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

Proof sketch. It suffices to show that MaxMatching is in non-uniform DynFO. The idea is to advise a dynamic program with the weighting functions $w^1, \ldots, w^{n^2}$ that assign weights such that for all graphs with n nodes there is a maximum matching with unique weight. The advice is given to the dynamic program via the initialization of the auxiliary relations. The program then computes the ranks for the matrices B_{G,w_i} and outputs the maximal such rank. We make this more precise in the following.

Recall that the weighting functions assign values of up to $4n$, and that therefore the determinant of each B_{G,w_i} can be of size up to $n!(2^{4n})^n \leq 2^{5n^2}$, and thus it is sufficient to maintain the rank of those matrices modulo up to $5n^2$ many primes, which are contained in the first n^3 numbers by the prime number theorem[11]. Such prime numbers and arithmetics in $\mathbb{Z}_p$ can be expressed with the help of triples over DOM; and as before we use single variables for denoting numbers $\leq n^3$. They represent three variables over DOM.

When inserting an edge (i, j) into the graph G, the (i, j)-entry of $B_{G,w}$ is set to $2^{w_{i,j}}$ for each of the weighting functions. Those values are too large to be encoded as tuples of elements. However, as we are interested in maintaining the rank of those matrices modulo small primes p only, it is sufficient to encode $2^{w_{i,j}}$ modulo p (for all such primes p). Those values are the advice to the dynamic program.

The dynamic program has an auxiliary relation W which is initialized with all tuples[12] (w, p, i, j, a) with $2^{w_{i,j}} = a$ modulo p.

The rank of a matrix $B_{G,w}$ for weighting function w can now be maintained by extending the relations B, C, R and Q from the proof of Theorem 2 by an additional argument for encoding the weighting (and also increasing the available numbers from n^2 to n^3). For example, we maintain a relation $\hat{B}$ that contains a tuple (w, p, i, j, a) if p is a prime number and, in the run of the dynamic algorithm

[11] We disregard small values of n as the query can be directly encoded with first-order formulas for such values.

[12] We note that such a tuple is 11-ary, as w, p and a represent triples.

for $B_{G,w}$ modulo p, the i-th vector of the basis has the j-entry a. Similarly for $\hat{C}$, $\hat{R}$ and $\hat{Q}$. All these relations can be updated as before, with the only difference that when an edge (i, j) is inserted, then its weight modulo the primes p is looked up in W, and used for the computations.

The relation $\hat{Q}$ stores the rank of $B_{G,w}$ for each weight W. From this relation the size of a maximum matching can be easily extracted by choosing the largest rank achieved by some weighting function. □

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 [17,5]. 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 seems worthwhile to reflect on the role of non-uniformity and isolation. The NL/poly = UL/poly [30] result seems to indicate that isolation helps in solving reachability by bringing in non-uniformity for general structures. [1] extends this idea to Matchings and [3,6,7,4] confirm this philosophy by eliminating non-uniformity in restricted structures such as planar/low genus graphs.

The situation after [5] was somewhat similar in the dynamic world. This work changes that world view by avoiding isolation altogether for Dynamic Reachability. As a bonus it yields a dynamic upper bound better than [5] had obtained through "traditional" methods. Not so surprisingly, maximum matching can also be maintained in DynFO though *non-uniformly*. We believe that these bounds can be translated to grid graphs using deterministic isolating weights [6] to show a DynFO upper bound.

At this point the magic spring seems to dry up - the non-uniformity in the case of general bipartite matching seems very hard to get rid of. The source of hardness seems to be in isolating a witness for a shortest directed path. In fact even given distance information[13] we do not know how to obtain a witness. Paradoxically this is a trivial problem in the bounded space world given a distance oracle.

The search for a uniform reachability witness maintenance algorithm seems to be the next question to ask. In fact we conjecture:

- A Reachability witness can be maintained in DynFO.
- A Shortest Path witness can be maintained in DynTC^0.

[13] Notice that it is known how to maintain distance in digraphs in DynTC^0[17] and undirected graphs in DynFO [15,25].

There are some further open questions related to the Reachability query itself. Altogether, our dynamic program for Reach uses 13-ary relations. It is known from [10] that Reach is not in DynFO with only unary auxiliary relations. It remains open whether Reach can be expressed with relations of lower arity, particularly, whether it is in binary DynFO. Another interesting question is whether Reachability can be maintained by even weaker update mechanisms, e.g. NC^0-updates. Lower bounds for this fragment are conceivable. Yet, even for the quantifier-free fragment of DynFO, which corresponds to restricted NC^0-updates, lower bounds are nontrivial. It is known that binary auxiliary relations are not sufficient to maintain Reachability in this fragment of DynFO [33]. We are grateful to the anonymous referee of ICALP 2015 who brought [13] to our attention.

Acknowledgements

We would like to thank William Hesse for stimulating and illuminating discussions. Further we thank Nils Vortmeier for proofreading. The first and the third authors were partially funded by a grant from Infosys Foundation. 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. Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *TOCT*, 1(1), 2009.
4. Samir Datta, Arjun Gopalan, Raghav Kulkarni, and Raghunath Tewari. Improved bounds for bipartite matching on surfaces. In *STACS*, pages 254–265, 2012.
5. Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In *ICALP (1)*, pages 356–367, 2014.
6. Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.
7. Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.*, 78(3):765–779, 2012.
8. Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *DBPL 1993*, pages 295–308, 1993.
9. 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.
10. 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.
11. Guozhu Dong and Rodney W. Topor. Incremental evaluation of datalog queries. In *Database Theory - ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, pages 282–296, 1992.

12. Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS*, pages 235–243, 1998.
13. Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. Dynamic matrix rank. *Theor. Comput. Sci.*, 410(41):4085–4093, 2009.
14. Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012.
15. Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT*, pages 236–248, 2012.
16. Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT*, pages 236–248, 2012.
17. William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003.
18. William Hesse and Neil Immerman. Complete problems for dynamic complexity classes. In *LICS*, page 313, 2002.
19. Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010.
20. Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.
21. Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
22. Stasys Jukna. *Extremal combinatorics*, volume 2. Springer, 2001.
23. Bastian Laubner. *The structure of graphs and new logics for the characterization of Polynomial Time*. PhD thesis, Humboldt University of Berlin, 2011.
24. László Lovász. On determinants, matchings, and random algorithms. In *FCT*, volume 79, pages 565–574, 1979.
25. Jenish C. Mehta. Dynamic Complexity of Planar 3-connected Graph Isomorphism. In *CSR*, page Accepted., 2014.
26. Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
27. Sushant Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *Journal of Computer and System Sciences*, 55(2):199–209, October 1997.
28. Michael O. Rabin and Vijay V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, 1989.
29. John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22(2):347–371, 1997.
30. Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.
31. Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *FOCS*, pages 509–517, 2004.
32. William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.
33. Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015.