Basic Research in Computer Science

Dynamic Normal Forms and Dynamic Characteristic Polynomial

Gudmund Skovbjerg Frandsen
Piotr Sankowski

BRICS Report Series **RS-08-2**

ISSN 0909-0878 **April 2008**

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
DK–8200 Aarhus N
Denmark

Telephone: +45 8942 9300
Telefax: +45 8942 5601
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/08/2/`

Dynamic Normal Forms and Dynamic Characteristic Polynomial

Gudmund Skovbjerg Frandsen[1] Piotr Sankowski[2]

[1] University of Aarhus, Denmark, gudmund@daimi.au.dk
[2] Warsaw University, Poland and University of Rome "La Sapienza", Italy, piotr.sankowski@gmail.com

Abstract. We present the first fully dynamic algorithm for computing the characteristic polynomial of a matrix. In the generic symmetric case our algorithm supports rank-one updates in $O(n^2 \log n)$ randomized time and queries in constant time, whereas in the general case the algorithm works in $O(n^2 k \log n)$ randomized time, where k is the number of invariant factors of the matrix. The algorithm is based on the first dynamic algorithm for computing normal forms of a matrix such as the Frobenius normal form or the tridiagonal symmetric form. The algorithm can be extended to solve the matrix eigenproblem with relative error 2^{-b} in additional $O(n \log^2 n \log b)$ time. Furthermore, it can be used to dynamically maintain the singular value decomposition (SVD) of a generic matrix. Together with the algorithm the hardness of the problem is studied. For the symmetric case we present an $\Omega(n^2)$ lower bound for rank-one updates and an $\Omega(n)$ lower bound for element updates.

Introduction. The computation of the *characteristic polynomial* (CP) of a matrix and the eigenproblem are two important problems in linear algebra and they find an enormous number of applications in mathematics, physics and computer science. Till now almost nothing about the dynamic complexity of these problems has been known. The CP problem is essentially equivalent to the computation of the Frobenius Normal Form (FNF), known also as rational canonical form. All of the efficient algorithms for CP are based on the FNF computation [1–4]. The fastest static algorithms for computing CP are either based on fast matrix multiplication and work in $\tilde{O}(n^\omega)$ time [4, 1] or they are so called black-box approaches working in $\tilde{O}(nm)$ time [2, 3], where m is the number of nonzero entries in the matrix. The latter bound holds only in the generic case, whereas the fastest general algorithm works in $O(\mu nm)$ [3], where μ is the number of distinct invariant factors of the matrix. All of these results have been obtained very recently. In this paper we are trying to understand the dynamic complexity of these fundamental problems by devising efficient algorithms and by proving matching lower bounds. Note, that in this paper and in all of the papers cited above, we study the arithmetic complexity of the problem, i.e., the notion of time is equivalent to the count of arithmetic operations and discrete control operations. More strictly speaking we work in the real RAM model, for details see [5].

In the first part of the paper we consider the problem of computing the normal form of a real (complex) $n \times n$ dynamic matrix A. We assume that the matrix

can be changed with use of rank-one updates, i.e., for two n dimensional vectors a and b we allow updates of the form $A := A + ab^T$. We want to dynamically compute matrices Q and F such that $A = Q^{-1}FQ$, where Q is the unitary similarity transformation and F is the normal form in question. The algorithm should support queries to F as well as vector queries to Q, i.e., given vector v it should be able to return Qv or $Q^{-1}v$. We present the following fully dynamic randomized algorithms for this problem:

- for generic symmetric matrices — an algorithm for tridiagonal normal form supporting updates in $O(n^2 \log n)$ worst-case time,
- for general matrices — an algorithm for Frobenius normal form (for definition see Section 1.1) supporting updates in $O(kn^2 \log n)$ worst-case time, where k is the number of invariant factors of the matrix.

The queries for F are answered in $O(1)$ time and the queries to Q in $O(n^2 \log n)$ worst-case time. After each update the algorithms can compute the characteristic polynomial explicitly and hence support queries for CP in constant time. These are the first known fully dynamic algorithms for the CP and normal form problems. Our results are based on a general result which can be applied to any normal form, under condition of availability of a static algorithm for computing the normal form of a sparse matrix. For the completeness of the presentation we have included the full algorithm for generic symmetric matrices, whereas the included algorithm for Frobenius normal form is the most universal result.

This algorithm can be extended to solve the dynamic eigenproblem, i.e., we are asked to maintain with relative error 2^{-b} the eigenvalues $\lambda_1, \ldots, \lambda_n$ and a matrix Q composed of eigenvectors. In generic case, our algorithm supports updates in $O(n \log^2 n \log b + n^2 \log n)$ worst-case time, queries to λ_i in constant time and vector queries to Q in $O(n^2 \log n)$ worst-case time. You should note that the relative error 2^{-b} is immanent even in the exact arithmetic model, i.e., the eigenvalues can only be computed approximately (for more details please see [6]).

Let A be a real (complex) $m \times n$ matrix, $m \geq n$. The singular value decomposition (SVD) for A consists in two orthogonal (unitary) matrices U and V and a diagonal matrix $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ with nonnegative real entries (the singular values) such that $A = U\Sigma V^T$. Usually the entries of Σ are sorted $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ and in that case Σ is unique. We define Σ_k to be $\mathrm{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0)$. The dynamic SVD problem considers maintaining the SVD under rank one updates and 2 query operations, one operation returns elements of Σ, another returns the k-rank approximation to A, i.e., given r and v return $U\Sigma_r V^T v$. Here, again the results are with a relative error 2^{-b}. Our algorithm for SVD supports updates in $O(n \log^2 n \log b + n^2 \log n)$ worst-case time in the generic case, queries to Σ in constant time and r-rank approximation query in $O(n^2 \log n)$ worst-case time.

Accompanying the above upper bounds, we provide some lower bounds for the problem of computing the characteristic polynomial. The lower bounds are formulated in the model of history dependent algebraic computation trees [7]. One should note that our algorithms for computing the CP fit into this model. We

use the technique developed and used by [7] for proving $\Omega(n)$ lower bounds for several dynamic matrix problems. The technique has been used later to prove an $\Omega(n)$ lower bound for the matrix rank problem [8]. Here, we significantly extend the technique to show an $\Omega(n^2)$ lower bound for the problem of computing the characteristic polynomial in the case of column updates. This is the first known result of this type. A column update can be realized with the use of one rank-one update. Additionally, we provide $\Omega(n)$ lower bounds for the CP problem in the case of element updates.

The paper is organized as follows. In the next subsection we motivate our study by reviewing possible applications within the scope of computer science. Nevertheless, note that the eigenproblem is THE method to study physical systems and our result could be applied to speed-up the physical computations in case when system parameters can be changed. In Section 1 we introduce the algorithms mentioned above. Section 2 includes the description of the obtained lower bound.

Applications and Earlier Work. Our result delivers a general framework for solving many problems that are based on the computation of matrix normal forms and on the solution of the matrix eigenproblem. Hence it generalizes a large number of problem specific solutions and can be directly applied to a broad spectrum of problems. Until now, it has been known how to dynamically compute the lowest coefficient of the CP, i.e., the determinant [9] and the rank of the matrix [8]. Our paper generalizes these two results as the CP can be used for both computing the determinant and in a rather simple way for computing the matrix rank [10, 11].

Our algorithms can be used to maintain dynamic information about the spectrum of a graph. Spectral graph theory has a large number of applications (see e.g. [12]) and delivers a way to compute numerous information about the graph. One of the possible applications is a dynamic testing of graph isomorphism, where one of the basic tools is a characteristic polynomial of the adjacency or Laplacian matrix [13].

Our algorithms for computing eigenvalues and eigenvectors can be used for both dynamically approximating the size of the graph partition and for finding good candidates for partition, e.g., the second smallest eigenvalue is related to the minimum partition size [14, 15]. There are several spectral methods for finding partitions and clusters in the graphs which can be used together with our algorithm [16, 17]. Clustering methods find application in image recognition and processing [16], where dynamic algorithms may be useful to process image changes.

Another direct application of our results is the dynamic maintenance of the stationary distribution of the finite Markov chain. For this problem slightly faster algorithms, working in $O(n^2)$ time, based on Sherman-Morison formula has been presented in [18–20]. However, our result is more general and can be used to check if the stationary distribution is unique or to compute the convergence time to stationary distribution by finding second largest eigenvalue [21], etc..

The SVD of a matrix may be directly used for finding the nearest matrix of a rank at most r by zeroing all singular values except the r largest [22]. For such a use only the leftmost r columns of U and V need to be known. Our concrete dynamic algorithm below allows application of the rank r approximation matrix to a vector in time $O(m^2 \log m)$ for any r. Indirectly dynamic SVD has many applications, e.g., for image analysis [23], in databases [24] and for data mining (recommender systems) [25].

In the case when addition of an entire vector or deletion of the last column of the matrix is allowed algorithms that take $O((m+n)\min(m,n))$ time per operation are known [26, 27]. Brand [25] has described an algorithm for rank one updates of the SVD (similar to our model) that takes time $O(mr + r^3)$ for maintaining a rank r approximation. Hence our solution improves these results as well.

1 Dynamic Characteristic Polynomial - Upper Bounds

In this section we show the algorithms for dynamically computing the characteristic polynomial, computing normal form and solving matrix eigenproblem. We show that the CP problem is strongly related to the static problem of computing the CP of a sparse or structured matrix. Standard methods for computing CPs transform the matrix to a normal form from which the characteristic polynomial can be easily computed. If the normal form has $O(n)$ non-zero entries and its CP can be computed in $O(n^2)$ time, it is called a *thin* normal form. For example, the Frobenious or the tridiagonal normal forms are thin. We assume, we are given a static algorithm that computes a thin normal form, a transition matrix and its inverse with use of $O(n)$ matrix-vector multiplications and $O(f(n))$ additional operations. We show how to convert this algorithm into a dynamic algorithm for computing a thin normal form supporting updates in $O(n^2 \log n + f(n))$ amortized time and queries in constant time. This result automatically implies a dynamic algorithm for computing the CP. Next we move to the application of this result and show implementations of this solution in the symmetric and general case. If the algorithm has some additional properties, we can turn it into a dynamic worst-case time algorithm. Finally we show how to extend the result to dynamically solve the matrix eigenproblem and SVD problem.

1.1 Amortized Bound

In the algorithm we keep the $n \times n$ matrix A over the field $\mathcal{F}$ in the following lazy form:

$$A = Q_0^{-1} Q_1^{-1} \dots Q_{k-1}^{-1} Q_k^{-1} T Q_k Q_{k-1} \dots Q_1 Q_0, \tag{1}$$

where $k \leq \lceil \log n \rceil$, the matrices Q_i, for $i = 1, \dots, n$, are some similarity transformations and T is a thin normal form. In each update we recompute the lazy form of the matrix and afterwards we compute its characteristic polynomial with use of the matrix T. We initialize the algorithm with the matrix A_0 and compute its normal form $A_0 = Q_0^{-1} T_0 Q_0$. Moreover, after n updates we reinitialize the

algorithm. Let us consider the sequence of t rank-one updates given by vectors a_i and b_i, for $i = 1, \ldots, t$, and let A_i denote the matrix after the i-th update, i.e.:

$$A_i = A_0 + \sum_{j=1}^{i} a_j b_j^T.$$

Let:

$$t = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_k}, \tag{2}$$

where $j_1 > j_2 > \ldots > j_k$. We require that the lazy form (1) fulfils the following:

$$A_{2^{j_1}+\ldots+2^{j_i}} = Q_0^{-1} Q_1^{-1} \ldots Q_{i-1}^{-1} Q_i^{-1} T_i Q_i Q_{i-1} \ldots Q_1 Q_0, \tag{3}$$

for $i = 1, \ldots, k$ and for T_i in the thin normal form. Now consider a new update numbered $t+1$. We have $t+1 = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_{k'}} + 2^{j'}$, for some $k' \leq k$ and $j' < j_{k'}$. Thus we have to compute a new lazy form fulfilling:

$$A + a_{t+1} b_{t+1}^T = Q_0^{-1} Q_1^{-1} \ldots Q_{k'}^{-1} {Q'}_{k'+1}^{-1} T Q'_{k'+1} Q_{k'} \ldots Q_1 Q_0. \tag{4}$$

Note that in order to recompute this form we have to discard the matrices $Q_{k'+1}, \ldots, Q_k$ and compute a matrix $Q'_{k'+1}$. Hence applying (3) we have:

$$A + a_{t+1} b_{t+1}^T = A_{2^{j_1}+\ldots+2^{j_{k'}}} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} a_j b_j^T =$$

$$= Q_0^{-1} Q_1^{-1} \ldots Q_{k'-1}^{-1} Q_{k'}^{-1} T_{k'} Q_{k'} Q_{k'-1} \ldots Q_1 Q_0 + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} a_j b_j^T =$$

$$= Q_0^{-1} \ldots Q_{k'}^{-1} \left(T_{k'} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} Q_{k'} \ldots Q_0 a_j b_j^T Q_0^{-1} \ldots Q_{k'}^{-1} \right) Q_{k'} \ldots Q_0.$$

Thus we have to compute the normal form of the matrix D_{t+1}:

$$D_{t+1} := T_{k'} + \sum_{j=2^{j_1}+\ldots+2^{j_{k'}}+1}^{2^{j_1}+\ldots+2^{j_{k'}}+2^{j'}} Q_{k'} \ldots Q_0 a_j b_j^T Q_0^{-1} \ldots Q_{k'}^{-1}. \tag{5}$$

Note that the vectors $a_{j,k'} = Q_{k'} \ldots Q_0 a_j$ and $b_{j,k'}^T = b_j^T Q_0^{-1} \ldots Q_{k'}^{-1}$, for $j \leq t$, are computed at the time when the algorithm was recomputing the lazy form after the j-th update. At the time of the j-th update k was greater than k' and so the matrices $Q_0, \ldots, Q_{k'}$ have not changed since then. Thus we only need to compute the vectors $a_{t+1,k'}$ and $b_{t+1,k'}$ when we are performing the $t+1$-th update — this takes $O(n^2 \log n)$ time. The multiplication of a vector by the matrix D_{t+1} takes $O(n 2^{j'})$ time. Hence for the computation of the normal form of D_{t+1} we need $O(n^2 2^{j'} + f(n))$ time.

Let us now compute the total cost of performing n updates:

– for the initialization of the lazy form we need $O(n^3 + f(n))$ time,

– for the computation of vectors $a_{j,i}$ and $b_{j,i}$ for $i = 1, \ldots, k$ we require $O(n \cdot n^2 \log n) = O(n^3 \log n)$ time,
– for the normal forms of D_i we need $O(nf(n) + \sum_{j=1}^{\lceil \log n \rceil} 2^{\lceil \log n \rceil - j} n^2 2^j) = O(n^3 \log n + nf(n))$ time, because we spend $O(n^2 2^j)$ time for computing the normal form $\frac{n}{2^j}$ times, namely every 2^j-th update.

Thus finally we get.

Theorem 1. *If there exists an algorithm for computing a given thin normal form, transition matrix and its inverse by performing $O(n)$ matrix-vector products and $O(f(n))$ additional operations then there exists a dynamic algorithm that maintains the characteristic polynomial and the thin normal form supporting rank one updates in $O(n^2 \log n + f(n))$ amortized time, queries to the normal form in constant time and vector queries to the transition matrix in $O(n^2 \log n)$ time.*

Generic Symmetric Case. Let us move to the implementations of Theorem 1 and let us for the moment assume that the $n \times n$ matrix A remains symmetric during the updates, i.e., we consider only updates of the form $A := A + aa^T$, where a is an arbitrary n dimensional vector. We want to compute a unitary matrix Q and a symmetric tridiagonal matrix T such that $A = QTQ^T$. The following is the result which can be obtained with the use of standard Lanczos method. For the completeness of the presentation the details of the proofs of the following theorems are included in Appendix A.

Theorem 2. *There exists an algorithm for computing a tridiagonal form of a symmetric generic matrix (when the characteristic polynomial equals the minimal polynomial) with use of n matrix-vector products and $O(n^2)$ additional operations. The algorithm is randomized and succeeds with high probability.*

Lemma 1. *The symmetric tridiagonal form is thin.*

Combining Theorem 1, Lemma 1 and Theorem 2 we get the $O(n^2 \log n)$ amortized updated time dynamic algorithm for computing the CP and the tridiagonal form of the generic symmetric matrix.

General Case. In the general case we use the following results due to Eberly [2], who showed how the Frobenius normal form of a sparse matrix can be computed. Frobenius normal form F_A of a matrix A is a block diagonal matrix with companion matrices of monic polynomials $f_1, \ldots, f_k$ on the diagonal, where f_i is divisible by f_{i+1}, for $1 \leq i \leq k-1$ and $VAV^{-1} = F_A$. The companion matrix of a monic polynomial $x^d + g_{d-1}x^{d-1} + \ldots + g_1 x + g_0 \in \mathcal{F}[x]$ is a $d \times d$ matrix defined as:

$$C_g = \begin{bmatrix} 0 & \ldots & 0 & -g_0 \\ 1 & \ldots & 0 & -g_1 \\ & \ddots & \vdots & \vdots \\ 0 & \ldots & 1 & -g_{d-1} \end{bmatrix}.$$

The polynomials $f_1, \ldots, f_k$ are *the invariant factors* of A and k is the number of invariant factors. We have $\chi_A(\lambda) = \prod_{i=1}^{k} f_k(\lambda)$ and so F_A is thin as it can be used to compute the CP in $O(n^2)$ time.

Theorem 3 (Eberly '00). *There exists an algorithm for computing Frobenius normal form F of the matrix A together with the transition matrix V and its inverse with use of $O(n)$ matrix-vector products and $O(kn^2+n^2\log^2 n)$ additional operations, where k is the number of invariant factors of A. The algorithm is randomized and may fail with arbitrarily small probability.*

Using the above theorem together with Theorem 1 we obtain the $O(kn^2 \log^2 n)$ amortized updated time dynamic algorithm for computing the CP and the Frobenius form of the matrix.

Remark 1. The time bound in the above theorem can be reduced to $O(kn^2 + n^2 \log n)$ by keeping the inverse of the transition matrix in the lazy form as given in Section 4.4 of [2]. Then the matrix-vector multiplication by the inverse can be carried out in $O(n^2)$ time. We skip the details due to page limitation of this extended abstract. The same holds for all the following results, i.e., with little effort one can always obtain an $O(kn^2 \log n)$ time algorithm instead of the $O(kn^2 \log^2 n)$ time algorithm. In the next theorems we use the $O(kn^2 + n^2 \log n)$ time bound, but postpone the details to the full version of this paper.

1.2 Worst-Case Bound

The algorithm presented in the previous section works in amortized time bound. Here we show how to modify it to work in worst-case time using rebuilding technique. However, as we keep a set of $\log n$ matrices we have to be very careful devising the rebuilding in order to guarantee the same worst-case time. Notice, that the standard technique, so called global rebuilding, in which we use two copies of the structure used alternately for answering queries, does not work here due to the multilevel recomputations. Here, we can only rebuild small parts of the structure, so one may call the used technique *local rebuilding*. Moreover due to non-uniqueness of the standard forms we also have to guarantee that the small recomputed parts remain consistent during the execution of the algorithm. In order to guarantee that the normal forms remain consistent we cannot discard transition matrices, but we have to multiply them. We cannot use standard or even fast matrix multiplication because it is too slow for our purposes. However, we can show that transition matrices can be multiplied faster without using classical matrix multiplication. All the details of the techniques used to prove the following theorem are included in Appendix B.

Theorem 4. *There exists an algorithm that:*

- *for the generic matrices maintains tridiagonal normal form and supports updates in $O(n^2 \log n)$ worst-case time,*
- *for the general matrices with k invariant factors maintains Frobenius normal form and supports updates in $O(kn^2 \log n)$ worst-case,*

the queries to CP and the normal form are supported in constant time, whereas vector queries to transition matrix are supported in $O(n^2 \log n)$ time.

1.3 Dynamic Matrix Eigenproblem

Theorem 4 presented in the previous section can be extended to solve the matrix eigenproblem.

Theorem 5. *There exists a dynamic algorithm for the matrix eigenproblem supporting rank one updates:*

- *in $O(n^2 \log n + n \log^2 n \log b)$ worst-case time for symmetric matrices,*
- *in $O(kn^2 \log n + n \log^2 n \log b)$ worst-case time for general matrices with k invariant factors.*

The computations are carried out with relative error 2^{-b}, the queries to the eigenvalues are answered in constant time and vector queries to eigenvector matrix in $O(n^2 \log n)$ worst-case time.

Proof. Note that the eigenvalues and eigenvectors of the maintained tridiagonal or Frobenius normal form F can be computed in $O(n^2 \log n + n \log^2 n \log b)$ time with use of the algorithm given by Pan and Chen [6]. The eigenvalues of F are of course the same as the eigenvalues of the maintained matrix. However, the eigenvectors have to be multiplied by $Q_0^{-1} Q_1^{-1} \dots Q_k^{-1} Q_0$ what takes $O(n^2 \log n)$ time for each eigenvector. □

1.4 Dynamic Singular Value Decomposition

Our algorithm for dynamic SVD is an application of the earlier results for dynamic eigenvalues and eigenvectors of symmetric matrices. The details and the proof of the following theorem are in Appendix C.

Theorem 6. *There exists a dynamic algorithm for SVD of a generic matrix supporting rank one updates in $O(n^2 \log n + n \log^2 n \log b)$ worst-case time. The computations are carried out with relative error 2^{-b} and the queries to the singular values are answered in constant time, whereas queries for r-rank approximation are answered in $O(n^2 \log n + nm)$ worst-case time.*

2 Dynamic Characteristic Polynomial - Lower Bounds

Problems considered. Let $s_i(A)$ or simply s_i denote the ith coefficient of the characteristic polynomial of the $n \times n$ matrix A over the field $\mathcal{F}$, i.e., $\chi_A(\lambda) = \det(\lambda I - A) = \lambda^n + \sum_{i=1}^{n} (-1)^i s_i \lambda^{n-i}$. We let our basic dynamic algebraic problem D associated with the characteristic polynomial consist in finding an efficient algorithm that after an initial preprocessing of $A = \{a_{ij}\}$ can handle operations $\texttt{change}_{ij}(v)$ that alters a_{ij} to v and operations $\texttt{query}_i$ that returns the current value of $s_i(A)$. To get stronger lower bounds, we also consider the problem D_i where we restrict ourselves to a single $\texttt{query}_i$ that may be automatically appended to all change operations that are thus required to maintain $s_i(A)$. We also consider the very restricted simple problem, D'_i, where we are only required to maintain information about whether $s_i(A)$ is zero or nonzero. All the above problems have variants that consider vector updates, i.e., changing an entire row and/or column of the matrix A instead of changing single entries of A. Similarly, all problems may be restricted to symmetric matrices, so change operations are paired symmetrically.

Model of computation. Our basic model of computation is the history dependent algebraic computation trees from [7]. A standard algebraic computation tree has computation nodes $+, \cdot, -, /$ and branching nodes (zero tests) [28]. For the field of real numbers, all continuous operations including square root used in the Lanczos algorithm, can be supported [5], and we also allow branching based on inequality tests. Each operation is assigned not one tree but many trees, namely one for each history where history means all discrete information obtained so far such as the sequence of operations applied earlier and the results of branching tests in earlier operations. The memory consists of variables holding field values that are preserved between operations. The variables may be written and read by the computation trees. The complexity of a solution is the maximal height of any tree in it. All our algorithms are within this basic model. We state explicitly, when our lower bounds are valid in a weaker model only (such as straightline programs).

Results. The following theorem is immediate from the lower bound for dynamic computation of the determinant [7].

Theorem 7. *Let the field $\mathcal{F}$ be infinite. The problem D has complexity $\Omega(n)$.*

If we allow the more general column updates, then the lower bound can be improved to $\Omega(n^2)$. Actually, this is a corollary to a stronger result we prove, namely a lower bound for maintaining whether a single coefficient is zero.

Theorem 8. *Let the field $\mathcal{F}$ contain the real numbers. Let $1 \leq l \leq n$.*

The problem D'_l has complexity $\Omega(\min(l, n-l))$ for symmetric matrices.

The problem D'_l has complexity $\Omega(\min(l, n-l)^2)$ for symmetric matrices when using vector updates.

The proof of this result is based on a strengthening of the lower bound for matrix rank from [8].

Theorem 9. *Let $\mathcal{F}$ be an algebraically closed field or the real numbers. Consider dynamic computation of* $\operatorname{rank}(M)$ *where $M \in \mathcal{F}^{(3n+1)^2}$, M is symmetric (vector updates must be paired into symmetric row-column updates) and* $\operatorname{rank}(M)$ *must remain one of $2n$ and $2n+2$. This problem has complexity at least $n^2/4$.*

Proof (of Theorem 9). The proof uses a reduction from matrix vector multiplication verification (MVMV), where the MVMV problem consists in verifying that $Mx = y$ for square matrix M and column vectors x and y. The MVMV problem was introduced in [8], where a lower bound was shown for algebraically closed fields and element updates. Our main contribution is to extend the lower bound for MVMV to be valid for real numbers and vector updates, combined with an observation that the reduction from MVMV need only use the rank of symmetric matrices. For details see Appendix D.

Proof (of Theorem 8). It is known that the rank of a symmetric real matrix is precisely the number of nonzero roots of its characteristic polynomial [10]. Hence, for a matrix M as given in the statement of Theorem 9, we may distinguish

between the two possible ranks simply by checking whether s_{2n+2} is zero. By embedding M in a larger matrix M_1 that has zeros elsewhere:

$$M_1 = \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix},$$

we have the lower bound $\Omega(l^2)$ on deciding whether s_l is zero for $n \times n$ matrix when $l \leq \frac{2}{3}n$. Similarly, by embedding M in the upper left corner of a larger matrix M_2 that has an identity matrix in the lower right corner:

$$M_2 = \begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix},$$

we have the lower bound $\Omega((n-l)^2)$ on deciding whether s_l is zero for $n \times n$ matrix when $l > \frac{2}{3}n$. Combining the two bounds, we get $\Omega(\min(l, n-l)^2)$ for all l. When adjusting the arguments towards single element updates rather than vector updates, one may similarly prove the lower bound $\Omega(\min(l, n-l))$ for all l. □

For the problem of maintaining the value of a coefficient rather than simply maintaining whether the coefficient is zero, we can prove slightly stronger lower bounds for element updates than those of Theorem 8 but partly in a more restrictive model (proof in Appendix E).

Theorem 10. *Let the field $\mathcal{F}$ be infinite.*

The problem D_l has a straightline solution of complexity $O(1)$ for $l = 1, 2$.

The problem D_l has complexity $\Omega(\max(l, n/l))$ for $l \geq 3$.

The problem D_l has complexity $\Omega(n)$ for $l \geq 3$, when the model of computation is restricted to history dependent straightline programs without division, i.e. using operations $+, -, \cdot$ only.

3 Conclusion and Open Problems

In this paper we have proven that several fundamental problems in linear algebra allow to construct fully dynamic algorithms. We were able to show almost square worst-case time randomized dynamic algorithms in the generic case for the problems of computing: characteristic polynomial, tridiagonal symmetric form, Frobenius normal form, eigenvalues, eigenvectors, singular value decomposition and polynomial evaluated at the matrix. What is more important, the algorithms are practically applicable, i.e., work in worst-case time and the constant hidden in big-O is rather small. Moreover, we have been able to prove strong lower bounds for the problems. Hence, we have presented an extensive study of the arithmetic complexity of the problem. Nevertheless, our results rise a question whether similar but numerically correct algorithms can be obtained. We have decided to keep this issue out of the scope of the paper due to its size limitations. The following question are left open as well.

– The computation of the determinant can be carried out in subquadratic time in the case of element updates [9]. Is it possible to get similar algorithms in the case of CP?
– Can the query complexity for the eigenvectors in the above algorithm be reduced from $O(n^2 \log n)$ time to $\tilde{O}(n)$ time? This is possible when we are willing to spend $O(n^{2.5})$ time on updates — details will be included in the full version of the paper.
– It would be consistent with Theorem 10 if $s_{\sqrt{n}}$ could be maintained by computation trees of complexity $\sqrt{n}$, so an open problem is to extend the $\Omega(n)$ lower bound for D_l ($l \geq 3$) to be valid for algebraic computation trees with division.
– It would be consistent with the above results if one could maintain singularity of a matrix with a dynamic algorithm of complexity $O(1)$. In particular Theorem 8 gives $\Omega(n)$ bounds on the complexity of D'_l only for the middle range of l values. This leads to the open problem of proving an $\Omega(n)$ bound for D'_l for general l.
– It is the first time an $\Omega(n^2)$ lower bound for a dynamic matrix problem has been obtained. Can it extended to work for dynamic transitive closure?

Acknowledgements. This work was partially supported by the EU within the 6th Framework Programme under contract 001907 "Dynamically Evolving, Large Scale Information Systems" (DELIS), by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL) and by the Polish Ministry of Science, grant KBN-1P03A01830.

References

1. Giesbrecht, M.: Nearly optimal algorithms for canonical matrix forms. SIAM Journal on Computing **24**(5) (1995) 948–969
2. Eberly, W.: Asymptotically efficient algorithms for the Frobenius form. Paper 723-26, Department of Computer Science, University of Calgary (2003)
3. Villard, G.: Computing the Frobenius normal form of a sparse matrix. In: The Third International Workshop on Computer Algebra in Scientific Computing, Springer-Verlag (2000) 395–407
4. Storjohann, A.: Deterministic computation of the frobenius form. In: FOCS. (2001) 368–377
5. Ben-Amram, A., Galil, Z.: On pointers versus addresses. J. Assoc. Comput. Mach **39** (1992) 617–648
6. Pan, V., Chen, Z.: The complexity of the matrix eigenproblem. In: STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1999) 507–516
7. Frandsen, G., Hansen, J., Miltersen, P.: Lower bounds for dynamic algebraic problems. Inform. and Comput. **171**(2) (2001) 333–349
8. Frandsen, P., Frandsen, G.: Dynamic matrix rank. In: ICALP. Volume 4051 of Lecture Notes in Comput. Sci. Springer, Berlin (2006) 395–406
9. Sankowski, P.: Dynamic Transitive Closure via Dynamic Matrix Inverse. In: FOCS. (2004) 509–517

10. Ibarra, O., Moran, S., Rosier, L.: A note on the parallel complexity of computing the rank of order n matrices. Inform. Process. Lett. **11**(4-5) (1980) 162
11. Mulmuley, K.: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. In: STOC, New York, NY, USA, ACM Press (1986) 338–339
12. Chung, F.R.K.: Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92). American Mathematical Society (February 1997)
13. Babai, L., Grigoryev, D., Mount, D.: Isomorphism of graphs with bounded eigenvalue multiplicity. In: STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing, New York, NY, USA (1982) 310–324
14. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal **23**(98) (1973) 289–305
15. Spielman, D., Teng, S.H.: Spectral partitioning works: Planar graphs and finite element meshes. In: FOCS. (1996) 96–105
16. Weiss, Y.: Segmentation using eigenvectors: A unifying view. In: ICCV (2). (1999) 975–982
17. Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Proceedings of Advances in Neural Information Processing Systems 14. (2001)
18. C.D. Meyer, J.S.: Updating finite markov chains by using techniques of group matrix inversion. J. Statist. Comput. Simulat. **11** (1980) 163–181
19. Funderlic, R.E., Plemmons, R.J.: Updating lu factorizations for computing stationary distributions. SIAM J. Algebraic Discrete Methods **7**(1) (1986) 30–42
20. Seneta, E.: Sensivity analysis, ergodicity coefficients, and rank-one updates for finite markov chains. In: Numerical Solutions of Markov Chains. (1991) 121–129
21. Diaconis, P., Stroock, D.: Geometric bounds for eigenvalues of markov chains. The Annals of Applied Probability **1**(1) (1991) 36–61
22. Golub, G.H., Van Loan, C.F.: Matrix computations. Third edn. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD (1996)
23. Chandrasekaran, S., Manjunath, B.S., Wang, Y.F., Winkeler, J., Zhang, H.: An eigenspace update algorithm for image analysis. Graph. Models Image Process. **59**(5) (1997) 321–332
24. Kanth, K., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. In: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, NY, USA (1998) 166–176
25. Brand, M.: Fast online svd revisions for lightweight recommender systems. In Barbará, D., Kamath, C., eds.: SDM, SIAM (2003)
26. Gu, M., Eisenstat, S.C.: A stable and fast algorithm for updating singular value decomposition. Technical Report YALE/DCS/TR-966, Yale University, New Haven, CT (1993)
27. Gu, M., Eisenstat, S.: Downdating the singular value decomposition. SIAM Journal on Matrix Analysis and Applications **16**(3) (1995) 793–810
28. Bürgisser, P., Clausen, M., Shokrollahi, M.: Algebraic complexity theory. Volume 315 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin (1997)
29. Bini, D., Pan, V.: Polynomial and Matrix Computations. Birkhäuser (1994)
30. Eberly, W., Kaltofen, E.: On randomized lanczos algorithms. In: ISSAC '97: Proceedings of the 1997 international symposium on Symbolic and algebraic computation, New York, NY, USA, ACM Press (1997) 176–183

A Proof of Theorem 2 and Lemma 1

Given a symmetric matrix A we want to compute an unitary matrix Q and a symmetric tridiagonal matrix:

$$T = \begin{bmatrix} \alpha_0 & \beta_0 & & 0 \\ \beta_0 & \alpha_1 & \ddots & \\ & \ddots & \ddots & \beta_{n-2} \\ 0 & & \beta_{n-2} & \alpha_{n-1} \end{bmatrix},$$

such that $A = QTQ^T$. This equation is equivalent to $AQ = QT$. Now if we denote by q_i the i-th column of Q, we can write:

$$Aq_j = \beta_{j-1}q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}, \text{ for } j = 0, \ldots, n-1, \tag{6}$$

where we assume that $\beta_{-1} = \beta_{n-1} = 0$. Having the vector q_0 we can compute the entries α_i, β_i and the vectors q_i by means of the Lanczos algorithm [29] as given by Algorithm 1.

Algorithm 1 The Standard Lanczos Method

1. let q_0 be a random n dimensional vector over $\mathcal{F}$,
2. set $\beta_{-1} = 0$, $j = 0$,
3. compute $\alpha_j = q_j^T A q_j$, $r_j = (A - \alpha_j I)q_j - \beta_{j-1}q_{j-1}$,
4. if $r_j \neq 0$,
   - then, set $\beta_j = \|r_j\|_2$ and $q_{j+1} = \frac{r_j}{\beta_j}$,
   - else, stop and output FAILED,
5. set $j = j + 1$,
6. if $j \leq n - 1$ go to Stage 2, else stop and output the entries of T and Q.

Note that in the Lanczos algorithm we perform n matrix-vector products and $O(n^2)$ additional operations.

Theorem 11 (Theorem 2). *Algorithm 1 computes a tridiagonal form of a symmetric matrix with use of n matrix-vector products and $O(n^2)$ additional operations. The algorithm is randomized and succeeds with probability at least $1 - \frac{n(n+1)}{|\mathcal{F}|}$.*

The proof of the lemma is rather standard and it can be carried out in the same way as the proof of Theorem 4.2 from [30]. Now we only need to show the following.

Lemma 2 (Lemma 1). *The symmetric tridiagonal form is thin.*

Proof. The tridiagonal form has only $3n - 2$ non-zero entries. We just need to show how to compute its characteristic polynomial. Let T_i denote the $i \times i$ leading principal submatrix of T. We can express its characteristic polynomial

$\chi_{T_i}(\lambda) = \det(\lambda I - T_i)$, for $i = 1, \ldots, n$, by the following recurrence relations with use of the Laplace rule:

$$\begin{aligned}
&\chi_{T_0}(\lambda) = 1,\\
&\chi_{T_1}(\lambda) = \lambda - \alpha_0,\\
&\chi_{T_{i+1}}(\lambda) = (\lambda - \alpha_i)\chi_{T_i}(\lambda) - \beta_{i-1}^2 \chi_{T_{i-1}}(\lambda), \text{ for } i = 1, 2, \ldots, n-1.
\end{aligned}$$

This recurrence can be easily solved in $O(n^2)$ time. □

B Proof of Theorem 4

In order to guarantee that the normal forms remain consistent we cannot discard transition matrices, but we have to multiply them. However, we cannot use standard matrix multiplication because it is too slow for our purposes. We need the following results.

Lemma 3. *Let $A = Q_0 T_A Q_0^T$, and $B = Q_0 Q_1 T_B Q_1^T Q_0^T$, where T_A and T_B are tridiagonal matrices. Then the matrix Q such that $Q = Q_0 Q_1$ can be computed with use of n matrix-vector products and $O(n^2)$ additional operations.*

Proof. Note, that $BQ = QT_B$ and so as a consequence Q is uniquely defined by T_B and q_0 by the following recurrence relation $q_{j+1} = \frac{1}{\beta_j}(Bq_j - \alpha_j q_j - \beta_{j-1} q_{j-1})$. We can compute q_0 from Q_0 and Q_1 in $O(n^2)$ time and then solve this recurrence with use of n matrix-vector multiplications and $O(n^2)$ additional operations. □

We also need similar result for the general case.

Lemma 4. *Let $A = V_0^{-1} T_A V_0$, and $B = V_0^{-1} V_1^{-1} T_B V_1 V_0$, where F_A and F_B are Frobenious matrices. Then the matrix V such that $V = V_1 V_0$ can be computed with use of n matrix-vector products and $O(kn^2)$ additional operations, where k is the number of invariant factors of B.*

Proof. Let $f_1, \ldots, f_k$ be the nontrivial invariant factors of B and let $d_i = \deg(f_i)$. Then the transition matrix V can be written as

$$V = [v_1, Bv_1, \ldots, B^{d_1 - 1} v_1, \ldots, v_k, Bv_k, \ldots, B^{d_k - 1} v_k],$$

for some vectors $v_1, \ldots, v_k$ (see eg. [2]). We can compute all v_i, for $1 \leq i \leq k$ from the product $V_1 V_0$ in $O(kn^2)$ time. Next the matrix V can be reconstructed with use of additional n matrix-vector products. □

When the algorithm for computing the normal form and the transition matrix can be used to multiply the transition matrices in the above way we say that the algorithm allows for *fast transition matrix multiplication.*

In the algorithm from Section 1.1 we have to discard the matrices $Q_{k'+1}, \ldots, Q_k$ and compute a matrix $Q'_{k'+1}$ (see (1) and (4)). Now, instead of discarding the matrices $Q_{k'+1}, \ldots, Q_k$ we use them in order to compute the matrix $Q'_{k'+1}$. We start by computing the matrix Q'_k such that:

$$A + a_{t+1} b_{t+1}^T = Q_0^{-1} Q_1^{-1} \ldots Q_{k-1}^{-1} Q_k^{-1} {Q'}_k^{-1} T Q'_k Q_k Q_{k-1} \ldots Q_1 Q_0,$$

for some normal form T. This requires $O(n^2 \log n + f(n))$ time (see (5)). We now have to compute a matrix $Q'_{k'+1}$ such that:

$$A + a_{t+1} b_{t+1}^T = Q_0^{-1} Q_1^{-1} \dots Q_{k-1}^{-1} Q_{k'}^{-1} {Q'}_{k'+1}^{-1} T Q'_{k'+1} Q_{k'} Q_{k-1} \dots Q_1 Q_0,$$

in other words the matrix $Q'_{k'+1}$ satisfies:

$$Q'_{k'+1} = Q'_k Q_k \cdot \ldots \cdot Q_{k'+1}. \tag{7}$$

We will not compute $Q'_{k'+1}$ immediately, but instead we keep it in a lazy form and recompute it part by part. First of all note that the matrix $Q'_{k'+1}$ represents $2^{j'}$ updates and it will not be needed to compute another matrix with (7) during the next $2^{j'}$ updates (see (2) and following equation for $t+1$). At that time it should be explicitly computed. However, till this time we keep it in the following lazy form:

$$Q'_{k'+1} = Q'_l Q_l \cdot \ldots \cdot Q_{k'+1}.$$

for $k \geq l \geq k'$. The matrices Q'_l and Q_l represent $2 \cdot 2^{j_l} = 2 \cdot 2^{k-l}$ updates. Using Lemma 3 or Lemma 4 we can compute their product in $O(2^{k-l} n^2)$ time. We run this computation in the background during the next 2^{k-l} updates. In such way we need only $O(n^2)$ additional time during each update to maintain the lazy form of $Q'_{k'+1}$. Moreover after the next $2^{j'}$ updates the matrix $Q'_{k'+1}$ will be computed explicitly.

Keeping the matrices Q_k in the lazy forms may increase by a factor of $\log n$ the time needed to compute the vectors $a_{j,k'}$ and $b_{j,k'}$. Nevertheless, this is not the case as the following lemma shows.

Lemma 5. *The total number of transition matrices kept in all lazy forms of Q_k is at most $\lceil \log n \rceil$.*

Proof. Consider the lazy form $A = Q_0^{-1} Q_1^{-1} \dots Q_{k-1}^{-1} Q_k^{-1} T Q_k Q_{k-1} \dots Q_1 Q_0$, where the matrix Q_i represents 2^{j_i} updates and we have $t = 2^{j_0} + \ldots + 2^{j_k}$ and $j_0 > j_1 > \ldots > j_k$. The lazy form of a matrix Q_i contains at the moment of initialization j_i matrices. However, from that time we have served $2^{j_{i+1}} + \ldots + 2^{j_k} \geq 2^{j_{i+1}}$ updates. Note that at this point the first j_{i+1} matrices in the lazy form of Q_i have been recomputed and are now represented by a single matrix. Hence after t updates the lazy form of Q_i contains at most $j_i - j_{i+1}$ matrices. The total number of matrices is now smaller than $j_k + \sum_{i=0}^{k-1} j_i - j_{i+1} = j_0 \leq \lceil \log n \rceil$. □

As a consequence we get the following theorem.

Theorem 12. *If there exists an algorithm for computing a given thin normal form, transition matrix and its inverse by performing $O(n)$ matrix-vector products and $O(f(n))$ additional operations that allows for fast transition matrix multiplication then there exists a dynamic algorithm that maintains the characteristic polynomial and this thin normal form of the matrix supporting rank one updates in $O(n^2 \log n + f(n))$ worst-case time, queries to the normal form in constant time and vector queries to transition matrix in $O(n^2 \log n)$ worst-case time.*

The above theorem can be used to obtain the main results of the paper.

Theorem 13 (Theorem 4). *There exists an algorithm that:*
- *for the symmetric matrices maintains tridiagonal normal form and supports updates in $O(n^2 \log n)$ worst-case time using,*
- *for the general matrices with k invariant factors maintains Frobenius normal form and supports updates in $O(kn^2 \log n)$ worst-case,*

the queries to CP and the normal form are supported in constant time, whereas vector queries to transition matrix are supported in $O(n^2 \log n)$ time.

Proof. For symmetric case the result is obtained from the previous theorem by combining it with Theorem 2, Lemma 1 and Lemma 3, whereas for the general case we need to use Theorem 3 and Lemma 4. □

C Proof of Theorem 6

Our algorithm for dynamic SVD is an application of the earlier results for dynamic eigenvalues and eigenvectors of symmetric matrices. Recall that we want to compute a diagonal matrix Σ and two unitary matrices U and V such that $A = U\Sigma V^T$. Note that the symmetric matrix $A^T A$ satisfies $V^T A^T A V = \Sigma^2$, which implies that the singular values of A are the square roots of the eigenvalues of $A^T A$, and V consists of eigenvectors of $A^T A$.

Assume that we use our dynamic algorithm for maintaining a tridiagonalization T of the symmetric $A^T A$ matrix. Also maintain the eigenvalues $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ (where $\lambda_1 \geq \cdots \geq \lambda_n$) and a corresponding orthonormal matrix E of eigenvectors for T according to Theorem 12 and Theorem 5, i.e. $T = E\Lambda E^T$. For each change in the symmetric generic matrix this can be done in $O(n^2 \log n + n \log^2 n \log b)$ time with relative error 2^{-b}. The algorithm works only in the generic case, because if the minimal polynomial of the matrix is not equal to the characteristic polynomial then the eigenvectors returned by Theorem 5 may not be orthogonal. Now consider a change $A := A + ab^T$, that gives raise to $O(1)$ symmetric changes to $A^T A$, namely $A^T A := A^T A + (b + A^T a)(b + A^T a)^T + (\sqrt{a^T a} b)(\sqrt{a^T a} b)^T - (A^T a)(A^T a)^T - bb^T$. Consider a query for Σ. Simply compute and return $\mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$, which can be done well within the time bound $O(n^2 \log n)$.

Consider the rank approximation query, i.e., given a vector v and r, we have to return $U\Sigma_r V^T v$. Assume the current lazy representation is

$$A^T A = Q_0^{-1} \cdots Q_k^{-1} T Q_k \cdots Q_0,$$

where $k \leq \log n$. Note that $V^T = E^T Q_k \cdots Q_0$, whereas

$$U = AV\Sigma^{-1} = AQ_0^{-1} \cdots Q_k^{-1} E \Sigma^{-1}.$$

Hence, we get:

$$U\Sigma_r V^T v = AQ_0^{-1} \cdots Q_k^{-1} E I_r E^T Q_k \cdots Q_0 v,$$

what can be computed in $O(n^2 \log n + nm)$ worst-case time.

Theorem 14 (Theorem 6). *There exists an dynamic algorithm for SVD of a generic matrix supporting rank one updates in $O(n^2 \log n + n \log^2 n \log b)$ worst-case time. The computations are carried out with relative error 2^{-b} and the queries to the singular values are answered in constant time, whereas queries for r-rank approximation are answered in $O(n^2 \log n + nm)$ worst-case time.*

D Proof of Theorem 9

First, observe that the proof of the lower bound for matrix vector multiplication verification (MVMV) in [8] that is shown for algebraically closed fields can also be made to work for the reals, when replacing the algebraic incompressibility argument with an argument from topology:

Theorem 15. *Any history dependent computation tree solution for dynamic evaluation of $MVMV(M, x, y)$ where $(M, x, y) \in \mathbf{R}^{n^2} \times \mathbf{R}^n \times \mathbf{R}^n$ has complexity at least $n/4$.*

Proof. Let a family of computation trees solving dynamic evaluation of $MVMV$ be given, and let the max depth of any computation tree representing a `change` be d. Recall that branching may be based on inequality tests (and not just zero tests), and we allow arbitrary binary continuous operations $\mathbf{R}^2 \mapsto \mathbf{R}$ in addition to arithmetic operations $+, \cdot, -, /$.

If we concatenate several change operations into a composite change, we may compose the associated computation trees into a larger tree by letting the root of a tree replace a leaf in a previous tree. Let in this way $P = P_1; P_2; P_3$ denote the computation tree for off-line $MVMV(M, x, y)$ that arises by concatenating changes in the following order (with no prior history, all inputs are initially zero) assuming input variables $M = \{m_{ij}\}$, $x = \{x_i\}$, and $y = \{y_j\}$.

$$\begin{aligned} P_1 &: \texttt{change}_1(m_{11}); \cdots ; \texttt{change}_{n^2}(m_{nn}); \\ P_2 &: \texttt{change}_{n^2+1}(x_1); \cdots ; \texttt{change}_{n^2+n}(x_n); \\ P_3 &: \texttt{change}_{n^2+n+1}(y_1); \cdots ; \texttt{change}_{n^2+2n}(y_n); \end{aligned}$$

Define a modified tree $P' = P_1; P_2; P_3'$ where

$$P_3' : \texttt{change}_{n^2+n+1}((Mx)_1); \cdots ; \texttt{change}_{n^2+2n}((Mx)_n);$$

Note that P' is essentially P pruned to contain only leaves labelled *true*. Given specific values for M, x the computation will follow a specific path through P'. Note that among possible computation paths, there will be at least one main path $\pi = \pi_1; \pi_2; \pi_3$ satisfying that there is a nonempty open subset $S \in \mathbf{R}^{n^2+n}$ such that all $M, x \in S$ will follow the path π. Here π_1 denotes the portion of the path running through P_1 etc. The path π can also be found in the tree P, since P' is essentially a pruning of P, though π will not be a main path in P.

We may find nonempty open subsets $S_1 \in \mathbf{R}^{n^2}$ and $S_2 \in \mathbf{R}^n$ such that $S_1 \times S_2 \subseteq S$. Let V be the set of the variables that are written by computation nodes on π_1 and read by computation and branching nodes on $\pi_2; \pi_3$. Let $\mathbf{v} \in$

$\mathbf{R}^{|V|}$ denote the contents of V after the execution of π_1 but before the execution of $\pi_2;\pi_3$. Clearly, $\mathbf{v}$ is a continuous function of M. Let $g : S_1 \mapsto \mathbf{R}^{|V|}$ denote that continuous function.

We will now argue that g is injective. Assume to the contrary that we can find specific matrices $M_1, M_2 \in S_1$ with $M_1 \neq M_2$ and $g(M_1) = g(M_2)$. Let $W_2 = \{x \mid M_1 x = M_2 x\}$, which is a subset of $\mathbf{R}^n$. Since S_2 is open (and nonempty), $S_2 \setminus W_2$ must be nonempty, and we may choose some $x_1 \in S_2 \setminus W_2$. When the computation tree P is applied to the input $(M_1, x_1, M_1 x_1)$ it will follow path π and compute *true* as it should. However, when P is applied to input $(M_2, x_1, M_1 x_1)$ it will also follow path π, since $g(M_1) = g(M_2)$, and therefore also answer *true*, which is incorrect. By contradiction, we have shown that g is injective.

Since S_1 is a nonempty open subset of $\mathbf{R}^{n^2}$, we may find an injective continuous function $g' : \mathbf{R}^{n^2} \mapsto S_1$, but then $g \circ g' : \mathbf{R}^{n^2} \mapsto \mathbf{R}^{|V|}$ is an injective continuous function, which by [5, theorem 10] implies that $|V| \geq n^2$. However, since the path $\pi_2;\pi_3$ contains at most $2dn$ computation and branching nodes each of which can read at most 2 variables, it follows that $4dn \geq |V|$, implying that $d \geq n/4$. □

Note also that in the above argument changes only play a role when concatenated corresponding to an entire column, i.e., a column update. The same is true for [8, theorem 3], therefore we have.

Theorem 16. *Let $\mathcal{F}$ be an algebraically closed field or the real numbers. Then any solution allowing column updates for dynamic evaluation of $MVMV(M, x, y)$ where $(M, x, y) \in \mathcal{F}^{n^2} \times \mathcal{F}^n \times \mathcal{F}^n$ has complexity at least $n^2/4$.*

Finally, we adjust the reduction of the MVMV problem to matrix rank [8] to work in our case.

Theorem 17 (Theorem 9). *Let $\mathcal{F}$ be an algebraically closed field or the real numbers. Consider dynamic computation of* $\mathrm{rank}(M)$ *where $M \in \mathcal{F}^{(3n+1)^2}$, M is symmetric (vector updates must be paired into symmetric row-column updates) and* $\mathrm{rank}(M)$ *must remain one of $2n$ and $2n+2$. This problem has complexity at least $n^2/4$.*

Proof. Given an instance $(M, x, y) \in \mathcal{F}^{n^2+n+n}$ of MVMV, create a $(2n) \times (n+1)$ matrix

$$N = \begin{bmatrix} I & x \\ M & y \end{bmatrix}.$$

Clearly, $\mathrm{rank}(N) \in \{n, n+1\}$ and $\mathrm{rank}(N) = n$ if and only if $Mx = y$.

From N create an instance $M' \in \mathcal{F}^{(3n+1)^2}$ of matrix rank, where:

$$M' = \begin{bmatrix} 0 & N^T \\ N & 0 \end{bmatrix}.$$

Clearly, M' is symmetric, $\mathrm{rank}(M') \in \{2n, 2n+2\}$ and $\mathrm{rank}(M') = 2n$ if and only if $Mx = y$. Since the change of a column vector in the input (M, x, y)

corresponds to a paired row and column change of M', we have reduced dynamic MVMV to the restricted version of dynamic matrix rank, and the wanted result is implied by Theorem 16. □

E Proof of Theorem 10

The lower bound for determinant implies a lower bound of $\Omega(l)$ for maintaining the l-th coefficient. This follows from the following reduction. If B is the $n \times n$ block matrix:

$$B = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix},$$

containing the $l \times l$ matrix A in the upper left corner and zeros elsewhere then $\det A$ is the l-th coefficient of the characteristic polynomial of B.

It is possible to prove a linear lower bound for s_3 using a reduction from known lower bounds. But before giving this reduction, let us argue the upper bounds for s_1, s_2. Using $s_1 = \sum_{i=1}^{n} a_{ii}$, it is straightforward to maintain s_1 in time $O(1)$. Using that:

$$\frac{\partial s_2}{\partial a_{ij}} = \begin{cases} -a_{ji} & \text{for } i \neq j, \\ s_1 - a_{ii} & \text{for } i = j, \end{cases}$$

one similarly obtains that s_2 can be maintained in time $O(1)$ per change of an entry in A.

Finally, let us prove the lower bound for s_3. By [7] there is a an $\Omega(n)$ lower bound for dynamic matrix multiplication. The following equality implies a similar lower bound on dynamic matrix squaring:

$$\begin{pmatrix} I & M & 0 \\ 0 & I & N \\ 0 & 0 & I \end{pmatrix}^2 = \begin{pmatrix} I & 2M & MN \\ 0 & I & 2N \\ 0 & 0 & I \end{pmatrix}.$$

We can construct a solution for dynamic matrix squaring from a solution for maintaining s_3 under changes of the matrix. Let $B = \{b_{ij}\}$ where $B = A^2$. Assume we have a data structure for maintaining s_3 under changes of A. We want to be able to answer queries to b_{ij} for $i \neq j$ as well (queries to b_{ii} are not needed in order to make the above reduction work). For $i \neq j$ we have:

$$\frac{\partial s_3}{\partial a_{ij}} = b_{ji} - a_{ji} s_1.$$

Let D_k be a data structure for maintaining s_k, then we may answer a query for b_{ji} $(i \neq j)$ as follows using operations on D_3 and D_1:

old := D_3-query;
D_3-change$_{ij}(a_{ij}+1)$;
new := D_3-query;
D_3-change$_{ij}(a_{ij}-1)$;
return new $-$ old $+ a_{ji} D_1$-query

Since operations on D_1 can be done in time $O(1)$ some operation of D_3 must have complexity $\Omega(n)$.

To prove the bound $\Omega(n/l)$ for D_l, we are going to make a reduction showing that we can build a dynamic algorithm for s_3 from $l-2$ instances of a dynamic algorithm for s_l (for increasingly larger matrices). Given $n \times n$ matrix A and some m, let $B^{(m)}$ be the $(n+m) \times (n+m)$ matrix that has A in the upper left corner, an $m \times m$ identity matrix in the lower left corner and otherwise zeros:

$$B^{(m)} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}.$$

Define $s_i^{(m)}$ to be the coefficients of $\chi_{B^{(m)}}$, i.e.,

$$\chi_{B^{(m)}}(\lambda) = \lambda^{n+m} + \sum_{i=1}^{n+m} (-1)^i s_i^{(m)} \lambda^{n+m-i}.$$

Note that $\chi_{B^{(m)}}(\lambda) = (\lambda-1)^m \chi_A(\lambda)$. Elementary use of binomial expansion will verify that:

$$\begin{pmatrix} s_l^{(l-3)} \\ s_l^{(l-4)} \\ \vdots \\ s_l^1 \\ s_l^0 \end{pmatrix} = \begin{pmatrix} \binom{l-3}{l-3} & \binom{l-3}{l-4} & \cdots & \binom{l-3}{1} & \binom{l-3}{0} \\ 0 & \binom{l-4}{l-4} & \cdots & \binom{l-4}{1} & \binom{l-4}{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \binom{1}{1} & \binom{1}{0} \\ 0 & 0 & \cdots & 0 & \binom{0}{0} \end{pmatrix} \begin{pmatrix} s_3 \\ s_4 \\ \vdots \\ s_{l-1} \\ s_l \end{pmatrix}.$$

Observe that the above matrix is upper triangular with all ones in the diagonal. Hence it is invertible over any field. Let $(d_3, \ldots, d_l)$ denote the first row in the inverse matrix. It follows that $s_3 = \sum_{i=3}^{l} d_i s_l^{(l-i)}$. The lower bound for D_3 implies a lower bound of $\Omega(n/l)$ for D_l.

We can make a more economic reduction of s_3 to s_l, viz. we can build a dynamic algorithm for s_3 that only uses one instance of a dynamic algorithm for s_l (for a larger matrix), though there is a snag. The reduction works only if we stick to a model of computation that excludes division and branching tests, i.e., change operations are implemented by straight line programs using operations $+, \cdot, -$ only.

For an indeterminate x and $n \times n$ matrix A, let $C(x)$ be the $(n+l-3) \times (n+l-3)$ matrix that has Ax in the upper left corner, an $(l-3) \times (l-3)$ identity

matrix in the lower left corner and otherwise zeros:

$$C(x) = \begin{bmatrix} Ax & 0 \\ 0 & I \end{bmatrix}.$$

Define s_{ij} to be the coefficients of $\chi_{C(x)}$, i.e.,

$$\chi_{C(x)}(\lambda) = \sum_{i=0}^{n+l-3} (-1)^i \, (\sum_{j=0}^{i} s_{ij} x^j) \, \lambda^{n+l-3-i}.$$

An elementary computation may verify that $s_{l3} = s_3$.

Given a dynamic program for s_l, we may run that program on $C(x)$. Memory variables now contain coefficient vectors for polynomials in $k[x]$, field arithmetic is replaced by arithmetic in $k[x]$ and one maintains the polynomial $\sum_{j=0}^{l} s_{lj} x^j$. If no division or branching tests are used then one may maintain the truncated polynomial $\sum_{j=0}^{3} s_{lj} x^j$ correctly even when restricting all intermediate computations to arithmetic in $k[x]$ modulo x^4. Each modular polynomial arithmetic operation requires only $O(1)$ operations in the field, and we can still read off $s_3 = s_{l3}$.

Summing up, when disallowing division and branching, the lower bound for D_3 implies a lower bound of $\Omega(n - l)$ for D_l. Combining with the lower bound $\Omega(l)$ proved earlier, we get the lower bound $\Omega(n)$ for D_l when $l \geq 3$ in the restricted model without division and branching.

Recent BRICS Report Series Publications

RS-08-2 **Gudmund Skovbjerg Frandsen and Piotr Sankowski.** ***Dynamic Normal Forms and Dynamic Characteristic Polynomial*****. April 2008. 21 pp. To appear in ICALP '08.**

RS-08-1 **Anders Møller. 2008.**

RS-07-18 **Jan Midtgaard.** ***Control-Flow Analysis of Functional Programs*****. December 2007. iii+38 pp.**

RS-07-17 **Luca Aceto, Willem Jan Fokkink, and Anna Ingólfsdóttir.** ***A Cancellation Theorem for 7BCCSP*****. December 2007. 30 pp.**

RS-07-16 **Olivier Danvy and Kevin Millikin.** ***On the Equivalence between Small-Step and Big-Step Abstract Machines: A Simple Application of Lightweight Fusion*****. November 2007. ii+11 pp. To appear in Information Processing Letters (extended version). Supersedes BRICS RS-07-8.**

RS-07-15 **Jooyong Lee.** ***A Case for Dynamic Reverse-code Generation*****. August 2007. ii+10 pp.**

RS-07-14 **Olivier Danvy and Michael Spivey.** ***On Barron and Strachey's Cartesian Product Function*****. July 2007. ii+14 pp.**

RS-07-13 **Martin Lange.** ***Temporal Logics Beyond Regularity*****. July 2007. 82 pp.**

RS-07-12 **Gerth Stølting Brodal, Rolf Fagerberg, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave.** ***Optimal Resilient Dynamic Dictionaries*****. July 2007.**

RS-07-11 **Luca Aceto and Anna Ingólfsdóttir.** ***The Saga of the Axiomatization of Parallel Composition*****. June 2007. 15 pp. To appear in the Proceedings of CONCUR 2007, the 18th International Conference on Concurrency Theory (Lisbon, Portugal, September 4–7, 2007), Lecture Notes in Computer Science, Springer-Verlag, 2007.**

RS-07-10 **Claus Brabrand, Robert Giegerich, and Anders Møller.** ***Analyzing Ambiguity of Context-Free Grammars*****. May 2007. 17 pp. Full version of paper presented at CIAA '07.**

RS-07-9 **Janus Dam Nielsen and Michael I. Schwartzbach.** ***The SMCL Language Specification*****. March 2007.**