

Generating a d -dimensional linear subspace efficiently

Raphael Yuster *

Abstract

We present an algorithm for computing a d -dimensional subspace of the row space of a matrix. For an $n \times n$ matrix A with m nonzero entries and with $\text{rank}(A) \geq d$ the algorithm generates a $d \times n$ matrix with full row rank and which is a subspace of $\text{Rows}(A)$. If $\text{rank}(A) < d$ the algorithm generates a $\text{rank}(A) \times n$ row-equivalent matrix. The running time of the algorithm is

$$O(\min\{n^{2-2/\omega} m^{1/\omega} d^{\omega-2+1/\omega}, n^2 d^{\omega-2}\})$$

where $\omega < 2.376$ is the matrix multiplication exponent. An immediate corollary of the algorithm is the construction of a row-reduced equivalent matrix of A , and hence the computation of $\text{rank}(A)$, in time

$$O(\min\{n^{2-2/\omega} m^{1/\omega} \text{rank}(A)^{\omega-2+1/\omega}, n^2 \text{rank}(A)^{\omega-2}\}).$$

We note that the running time is sub-quadratic if $d < (n^2/m)^{0.528}$.

1 Introduction

Computing the rank of a matrix, and, more generally, generating a row-reduced equivalent matrix is one of the most basic computational algebraic problems and considerable effort is devoted to obtaining efficient algorithms for these tasks (see, e.g. [5]). One approach is naturally through Gaussian elimination. An old result of Hopcroft and Bunch [1] asserts that Gaussian elimination of a matrix requires asymptotically the same number of algebraic operations needed for matrix multiplication. It follows, in particular, that the algebraic complexity of rank computation of an $n \times n$ matrix over a field is $O(n^\omega)$ where $\omega < 2.376$ is the *matrix multiplication exponent* [2]. In fact, no faster algorithm is known for general $n \times n$ matrices.

If A is sparse and has only $m \ll n^2$ non-zero entries, it seems likely at first glance that computing a row-reduced matrix should be easier. Unfortunately, it is generally not known to be the case. This can be attributed to the negative result of Yannakakis [9] who proved that controlling the number of fill-ins (entries that were originally zero and become non-zero during the Gaussian elimination process due to pivoting) is an NP-Hard problem. The situation is somewhat better if we allow randomness. A seminal result of

Wiedemann [8] implies, in particular an $O(nm)$ Monte Carlo algorithm for solving a sparse linear system over a field. In particular the rank can be computed in $O(mn)$ time which is faster than Gaussian elimination if $m = o(n^{\omega-1})$. A randomized Las Vegas algorithm for computing the null-space basis (and hence the rank) of a matrix was obtained by Eberly et al [3]. The expected running time of their algorithm in the case where $m = \Theta(n)$ is $O(n^{3-1/(\omega-1)}) \leq O(n^{2.28})$.

The row-reduced matrix of a matrix A has $\text{rank}(A)$ rows. In particular, any set of d rows (of the row reduced matrix) where $d \leq \text{rank}(A)$ spans a d -dimensional subspace of $\text{Rows}(A)$ (the row space of A). The main result of this paper shows that generating a d -dimensional subspace of $\text{Rows}(A)$ requires (in some cases significantly) less time than just computing the whole row-reduced matrix. In particular our result shows that if $\text{Rank}(A)$ is relatively small then a row-reduced matrix can be found quickly. Another consequence is that queries of the form $\text{rank}(A) > d$ can be answered much faster (depending on d) than just computing the rank. Our main result is the following.

THEOREM 1.1. *Let A be an $n \times n$ matrix over an arbitrary field, containing m nonzero entries, and let $d \leq n$ be a positive integer. There is an algorithm that computes a $d \times n$ matrix with full row rank and which is a subspace of $\text{Rows}(A)$. If $\text{rank}(A) < d$ the algorithm generates a corresponding $\text{rank}(A) \times n$ row-reduced matrix. The running time of the algorithm is*

$$O(\min\{n^{2-2/\omega} m^{1/\omega} d^{\omega-2+1/\omega}, n^2 d^{\omega-2}\}).$$

As usual in algorithms that manipulate matrices, each algebraic operation in the field (addition, subtraction, multiplication, and division) has unit cost. If the field is finite and its number of elements is polynomial in n , then this also amounts to actual bit complexity (up to logarithmic factors).

An obvious consequence of Theorem 1.1 is that it can be used to answer queries of the form $\text{rank}(A) > d$ in the claimed running time. Another easy corollary of Theorem 1.1 is that a row-reduced matrix of A , and hence $\text{rank}(A)$, can be computed in time which is a function of the rank. This is achieved, say, by applying Theorem 1.1 only $O(\log(n))$ times with different values

*Department of Mathematics, University of Haifa, Haifa 31905, Israel. E-mail: raphy@math.haifa.ac.il

of d , binary searching for the correct rank. We thus have the following.

COROLLARY 1.1. *Let A be an $n \times n$ matrix over an arbitrary field, with m nonzero entries. There is an algorithm that computes a row-reduced matrix of A in time*

$$O(\min\{n^{2-2/\omega} m^{1/\omega} \text{rank}(A)^{\omega-2+1/\omega}, n^2 \text{rank}(A)^{\omega-2}\}).$$

If the matrix A is dense, say with $m = \Theta(n^2)$ then the running time in theorem 1.1 is $O(n^2 d^{\omega-2}) \leq O(n^2 d^{0.376})$. For every $d = n^{1-\epsilon}$ this is faster than just performing Gaussian elimination from scratch. But even more efficiency is obtained when the matrix is sparse. For any $m = n^{2-\epsilon}$ we see that for sufficiently small values of d one can do better than $O(n^2 d^{0.376})$ and, in fact, for even smaller values of d the running time is even better than $O(n^2)$ (as usual, it is assumed that the input matrix is given in a sparse representation). More precisely, if

$$d < \left(\frac{n^2}{m}\right)^{0.528} < \left(\frac{n^2}{m}\right)^{\frac{1}{\omega^2-2\omega+1}}$$

then the running time becomes sub-quadratic and if $d < n^2/m$ the running time is

$$O\left(n^2 d^{\omega-2} \left(\frac{n^2}{md}\right)^{-\frac{1}{\omega}}\right).$$

It is interesting to note that most, if not all, known matrix multiplication based algorithms do not exploit sparseness beyond some threshold of the form m^β where β is a constant strictly less than 1 (e.g. [10]) while the algorithm of Theorem 1.1 exploits sparseness already when $m = n^{2-\epsilon}$ for any ϵ .

The proof of Theorem 1.1 is composed of two parts. The first part is a procedure that reduces the problem of computing a d -dimensional subspace of the row space of a sparse matrix to the problem of computing a d -dimensional subspace of the row space of a dense, but smaller, *rectangular* matrix. The second part computes a d -dimensional subspace of the row space of a (possibly dense) rectangular matrix by repeatedly filtering out non-essential rows and replacing other rows with an equivalent set of rows spanning the same subspace. This part essentially uses the fact that matrices can be viewed as linear matroids.

2 Reducing sparse matrices to smaller dense rectangular matrices

We need the following result of Hopcroft and Bunch [1] asserting that Gaussian elimination of a matrix requires

asymptotically the same number of algebraic operations needed for matrix multiplication. Recall that Gaussian elimination of a matrix B produces a reduced row-equivalent matrix. That is, a matrix with the same number of columns but with only $\text{rank}(B)$ rows, and with full row rank.

LEMMA 2.1. *Let B be an $\ell \times n$ rectangular matrix over an arbitrary field. Then Gaussian elimination of B can be computed using $O(n\ell^{\omega-1})$ algebraic operations.*

We note that by using fast *rectangular* matrix multiplication [4] it is possible to improve the exponent a bit in the case where ℓ is significantly smaller than n . However, the improvement is negligible in our case and so we do not consider it further.

For the rest of this section A is an $n \times n$ matrix over an arbitrary field, containing m non-zero entries, and d is a positive integer not larger than n^2/m . The element of A in row i and column j is denoted by a_{ij} . We assume that the matrix is given with its sparse representation. That is, there are n linked lists $R(i)$ for $i = 1, \dots, n$, where the i 'th list contains items representing the non-zero elements of row i , sorted according to column index. Each item is therefore of the form (j, a_{ij}) . Notice that it is straightforward to create the corresponding column lists $C(j)$ for $j = 1, \dots, n$ in linear $O(m)$ time (we may assume that $m \geq n$ otherwise an entire row and column could be deleted).

LEMMA 2.2. *Let A be an $n \times n$ matrix over an arbitrary field, containing m non-zero entries, and let d be a positive integer not larger than n^2/m . For any s with $d \leq s \leq n$ there is an algorithm that constructs a matrix L with at most $d\lceil n/s \rceil$ rows and n columns with the following properties. If $\text{rank}(A) < d$ then L is row-equivalent to A . If $\text{rank}(A) \geq d$ then $\text{Rows}(L)$ is a subspace of $\text{Rows}(A)$ and $\text{Rows}(L)$ has dimension at least d . The running time of the algorithm is $O(ms^{\omega-1})$.*

Proof. We partition the rows of A into n/s rectangular matrices, each with s rows and n columns (we ignore rounding issues as these have no effect on the asymptotic nature of the result). Denote the rectangular matrices by A_1, \dots, A_t where $t = n/s$. Thus, A_1 consists of the first s rows of A , A_2 consists of the next s rows of A , and so on. Clearly, a sparse representation of A_i is obtained by just selecting the corresponding s row lists. Notice also that all the sparse column lists of all the A_i 's can be generated from the column lists of A in total time $O(m)$.

Let c_i denote the number of non-zero columns of

A_i . Clearly,

$$(2.1) \quad \sum_{i=1}^t c_i \leq m .$$

We can compact the A_i by just discarding the zero columns of A_i , thereby obtaining a matrix B_i with s rows and c_i columns. We do need to record the indices of the zero columns, or, equivalently, let $c_i(j)$ be the index of the j 'th non-zero column of A_i . That is, column j of B_i is column $c_i(j)$ of A_i .

Having constructed the B_i 's, we now perform Gaussian elimination on each of them, thus obtaining reduced equivalent matrices C_i with c_i columns and with $\text{rank}(B_i) = \text{rank}(A_i)$ rows for $i = 1, \dots, t$. By Lemma 2.1 the time required to construct C_i is $O(c_i s^{\omega-1})$. Thus, by (2.1), the overall running time required to construct all the C_i is

$$(2.2) \quad O(m s^{\omega-1}) .$$

Let D_i be the matrix obtained from C_i by taking the first d rows of C_i . If C_i has less than d rows (this happens if $\text{rank}(A_i) < d$) then just let $D_i = C_i$. Expanding D_i by plugging in the zero columns of A_i in the appropriate locations (using the recorded values $c_i(j)$) we obtain a matrix L_i with n columns. Now, if $\text{rank}(A_i) < d$ then L_i is row-equivalent to A_i (it spans the same row space as A_i). Otherwise, L_i is a $d \times n$ matrix spanning a d -dimensional subspace of $\text{Rows}(A_i)$.

Finally, take L to be the union of all the L_i . Notice that L has at most $dt = dn/s$ rows and n columns. Furthermore, if $\text{rank}(A) < d$ then L is row-equivalent to A , as each L_i must be row-equivalent to each A_i . Otherwise, if $\text{rank}(A) \geq d$ then either L is row-equivalent to A or else $\text{Rows}(L)$ is a subspace of $\text{Rows}(A)$ with dimension *at least* d . As the most time consuming operation of the algorithm is (2.2), the lemma follows.

3 Proof of the main result

LEMMA 3.1. *Let L be an $\ell \times n$ matrix over an arbitrary field and let d be a positive integer not greater than $\ell/2$. There is an algorithm that constructs a matrix L' with at most $\ell/2$ rows and n columns with the following properties. If $\text{rank}(L) < d$ then L' is row-equivalent to L . If $\text{rank}(L) \geq d$ then $\text{Rows}(L')$ is a subspace of $\text{Rows}(L)$ and $\text{Rows}(L')$ has dimension at least d . The running time of the algorithm is $O(\ell n d^{\omega-2})$.*

Proof. We partition the rows of L into $t = \lfloor \ell/2d \rfloor$ rectangular matrices A_1, \dots, A_t , each with $2d$ rows and n columns (the last matrix A_t may have more than $2d$ rows but less than $4d$ rows). Thus, A_1 consists of the first $2d$ rows of L , A_2 consists of the next $2d$ rows of L ,

and so on. We perform Gaussian elimination on each of the A_i thus obtaining reduced equivalent matrices C_i with n columns and with $\text{rank}(A_i)$ rows for $i = 1, \dots, t$. By Lemma 2.1 the time required to construct C_i is $O(n d^{\omega-1})$. Thus, the overall running time required to construct all the C_i is

$$(3.3) \quad O(t n d^{\omega-1}) = O(\ell n d^{\omega-2}) .$$

Let L_i be the matrix obtained from C_i by taking the first d rows of C_i . If C_i has less than d rows (this happens if $\text{rank}(A_i) < d$) then just let $L_i = C_i$. Now, if $\text{rank}(A_i) < d$ then L_i is row-equivalent to A_i . Otherwise, L_i is a $d \times n$ matrix spanning a d -dimensional subspace of $\text{Rows}(A_i)$. Finally, take L' to be the union of all the L_i . Notice that L' has at most $dt \leq \ell/2$ rows and n columns. Furthermore, if $\text{rank}(L) < d$ then L' is row-equivalent to L , as each L_i must be row-equivalent to each A_i . Otherwise, if $\text{rank}(L) \geq d$ then either L' is row-equivalent to L or else $\text{Rows}(L')$ is a subspace of $\text{Rows}(L)$ with dimension *at least* d . As the most time consuming operation of the algorithm is (3.3), the lemma follows.

Proof of Theorem 1.1: We are given an $n \times n$ matrix A containing m nonzero entries, and a positive integer d . If $d < n^2/m$ we apply Lemma 2.2 with a value of s between d and n that will be chosen later after optimization. The result of this application is a matrix L_0 with n column and $\ell_0 \leq d \lfloor n/s \rfloor$ columns satisfying the conditions of the lemma. If $d \geq n^2/m$ we do not apply Lemma 2.2. In this case we simply set $L_0 = A$ and $\ell_0 = n$. Thus, we can now assume that in any case, L_0 is an $\ell_0 \times n$ matrix so that if $\text{rank}(A) < d$ then L_0 is row-equivalent to A and if $\text{rank}(A) \geq d$ then $\text{Rows}(L_0)$ is a subspace of $\text{Rows}(A)$ and $\text{Rows}(L_0)$ has dimension *at least* d .

We now check if $d \leq \ell_0/2$. If so, we apply Lemma 3.1 to L_0 and obtain a matrix L_1 with $\ell_1 \leq \ell_0/2$ rows and n columns with the properties guaranteed by the lemma. We repeatedly apply Lemma 3.1 to L_i as long as $d \leq \ell_i/2$ and obtain the next matrix L_{i+1} . We halt when $d > \ell_i/2$. After halting, we perform a final Gaussian elimination to L_i and obtain the matrix B . If B has more than d rows, the output of the algorithm is just the first d rows of B . Otherwise, the output of the algorithm is B .

The correctness of the algorithm follows from two facts. At any stage of the algorithm, the rows of the present matrix L_i form a subspace of $\text{Rows}(A)$, and hence also at the end $\text{Rows}(B)$ is a subspace of $\text{Rows}(A)$. Thus, if B has at least d rows then the final output is just a d -dimensional subspace of $\text{Rows}(A)$. On the other hand, if $\text{Rank}(A) < d$ then each of the L_i , as well as the final B , is row-equivalent to A .

For the running time of the algorithm, consider first the case when $d \geq n^2/\ell$. In this case $\ell_0 = n$ and the application of Lemma 3.1 on L_i requires $O(\frac{1}{2^i}n^2d^{\omega-2})$ time. After $O(\log n)$ applications we finally compute B in $O(nd^{\omega-1})$ time by Lemma 2.1. Thus, the overall running time is $O(n^2d^{\omega-2})$, as required.

Consider next the case when $d < n^2/\ell$. The application of Lemma 2.2 that constructs L_0 requires $O(ms^{\omega-1})$ time. As $\ell_0 \leq d\lfloor n/s \rfloor$, the application of Lemma 3.1 on L_i requires $O(\frac{1}{s^{2^i}}n^2d^{\omega-1})$ time. After $O(\log n)$ applications we finally compute B in $O(nd^{\omega-1})$ time by Lemma 2.1. Thus, the overall running time is

$$O(ms^{\omega-1} + \frac{1}{s}n^2d^{\omega-1}).$$

Choosing the optimal value for s which is $s = (d^{\omega-1}n^2/m)^{1/\omega}$ (notice that indeed $s \geq d$ for this choice) we obtain that the overall running time in this case is $O(n^{2-2/\omega}m^{1/\omega}d^{\omega-2+1/\omega})$, as required. ■

4 Concluding remarks and open problems

- Naturally, it would be interesting to improve the time complexity of Theorem 1.1. A possible approach would be to make sure that the matrix L in Lemma 2.2 is also relatively sparse, assuming that A is such. We know that such an L exists (after all, there trivially exist d rows of A that form a d -dimensional subspace of $Rows(A)$), however we do not yet know how to find such an L . A possibly easier task is to improve the running time in Theorem 1.1 if we just settle for the seemingly easier problem of answering the query “ $rank(A) > d$?”.
- An example of an interesting application of our main result is for the maximum matching problem in graphs. Given an n -vertex undirected graph with m edges, Lovász has shown that computing the exact cardinality of a maximum matching with high probability, amounts to computing the rank of an $n \times n$ matrix with $O(m)$ non-zero entries over a finite field whose number of elements is polynomial in n . Hence, this leads to an $O(n^\omega)$ randomized algorithm for computing the cardinality of a maximum matching [6].

Now, suppose we just want to know, say, if there is a matching of cardinality at least d ? Then, we can use the algorithm of Theorem 1.1 to answer this question. In some cases, this leads to the fastest known method for this task. For example, suppose that $m = n^{\omega-1/2}$ and $d = n^\alpha$ where $\alpha \in (\omega - 1/2, 2)$ (to avoid trivialities we must assume that $m = o(nd)$ otherwise there trivially exist d independent edges). In this case the running

time is $O(n^{2+\alpha(\omega-2)})$. This should be compared to the $O(n^\omega)$ algorithm from [6], or to the $O(n^{1/2}m)$ algorithm of Michali and Vazirani [7] (the fastest known algorithm for maximum matching in terms of n and m for sufficiently sparse graphs) which, in this case, also takes $O(n^\omega)$ time.

References

- [1] J. Bunch and J. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28:231–236, 1974.
- [2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.
- [3] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computations using efficient block projections. *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)* ACM Press, 2007.
- [4] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.
- [5] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 378:91–130, 2004.
- [6] L. Lovász. On determinants, matchings, and random algorithms. In: *Fundamentals of computation theory, Vol. 2 pages 565–574*, Akademie-Verlag, Berlin, 1979.
- [7] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science (FOCS)*, 17–27, 1980.
- [8] D.H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.
- [9] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.
- [10] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1:2–13, 2005.