Algorithms for Fast Linear System Solving and Rank Profile Computation

by

Shiyun Yang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We give randomized algorithms for linear algebra problems concerning an input matrix $A \in \mathsf{K}^{n\times m}$ over a field K. We give an algorithm that simultaneously computes the row and column rank profiles of A in $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$ field operations in K, where r is the rank of A and $|A|$ denotes the number of nonzero entries of A. Here, the $o(1)$ in our cost estimates captures some missing $\log n$ and $\log m$ factors. The rank profiles algorithm is randomized of the Monte Carlo type: the correct answer will be returned with probability at least $1/2$. Given $b \in \mathsf{K}^{n\times 1}$, we give an algorithm that either computes a particular solution vector $x \in \mathsf{K}^{m\times 1}$ to the system $Ax = b$, or produces an inconsistency certificate vector $u \in \mathsf{K}^{1\times n}$ such that $uA = 0$ and $ub \neq 0$. The linear solver examines at most $r + 1$ rows and r columns of A and has running time $2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)}$ field operations in K, where $|R|$ and $|C|$ are the number of nonzero entries in the rows and columns, respectively, that are examined. The solver is randomized of the Las Vegas type: an incorrect result is never returned, but the algorithm may report FAIL with probability at most $1/2$. These cost estimates are achieved by making use of a novel randomized online data structure for the detection of linearly independent rows and columns.

The leading term $2r^3$ in the cost estimate $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$ of our rank profile algorithm arises from our use of an iterative algorithm to compute, for $s = 1, 2, \ldots, r$, the inverse of the leading principal $s \times s$ submatrix B_s of an $r \times r$ matrix B that has generic rank profile, and whose rows are given from first to last, one at a time, for $s = 1, 2, \ldots, r$. These inverses are used to compute a sequence of subsystem solutions $B_s^{-1}b_s$ for $s = 1, 2, \ldots, r$, where $b_s \in \mathsf{K}^{s\times 1}$ is the leading subvector of $b \in \mathsf{K}^{r\times 1}$. We give a relaxed algorithm that computes the sequence $B_1^{-1}b_1,\ B_2^{-1}b_2, \ldots, B_r^{-1}b_r$ in an online fashion in time $O(r^\omega)$, effectively allowing matrix multiplication to be incorporated into our rank profile algorithm. Together with a Toeplitz preconditioner, we can compute the row rank profile of a full column rank matrix A in time $(r^\omega + |A|)^{1+o(1)}$. Combined with Cheung, Kwok and Lau's (2013) algorithm for computing a maximal rank subset of linearly independent columns, this gives a Monte Carlo algorithm that computes the row rank profile of A in time $(r^\omega + |A|)^{1+o(1)}$.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Prof. Arne Storjohann for the invaluable ideas, comments, advice and constant encouragement. I could not have met a better supervisor. I would also like to thank Prof. George Labahn, Prof. Jeffrey O. Shallit and Prof. Romain Lebreton for their feedback in the preparation of this thesis. Finally, I want to thank all the friends I met from the Symbolic Computation Group. Finally, I would like to thank my parents for their endless love and support.

Table of Contents

List of Figures

Chapter 1

Introduction

In computational exact linear algebra, a classical problem is to transform a matrix over a finite field to row echelon form. A matrix is in row echelon form if

- zero rows, if any, lie below rows having a nonzero element,
- the first nonzero element in each nonzero row — *the pivot* — is in a column to the right of pivots in any previous rows, and
- all entries in a column below a pivot entry are zero.

For example, consider a matrix $A \in \mathsf{K}^{6\times 9}$ over a field K. A possible shape for the row echelon form of A is

$$\begin{bmatrix} \circledast & * & * & * & * & * & * & * & * \\ & & \circledast & * & * & * & * & * & * \\ & & & \circledast & * & * & * & * & * \\ & & & & & & \circledast & * & * \\ & & & & & & & & \circledast \\ & & & & & & & & \end{bmatrix}, \tag{1.1}$$

where $*$ indicates arbitrary elements of K, $\circledast$ indicates the nonzero pivots, and necessarily zero entries are simply left blank. Similarly, the column echelon form is just the transpose

situation. A possible shape for the column echelon form of A is

$$\begin{bmatrix} \circledast & & & & & \\ * & & & & & \\ * & \circledast & & & & \\ * & * & \circledast & & & \\ * & * & * & \circledast & & \\ * & * & * & * & \circledast & \end{bmatrix}. \tag{1.2}$$

Recall that transformation to row echelon form is accomplished using elementary row operations:

(i) interchanging two rows;

(ii) adding a multiple of one row to a different row;

(iii) multiplying a row by a nonzero element of K.

Algorithm 1 recalls the classical Gaussian elimination algorithm to transform a matrix to row echelon form. Line 7 performs operations of type (i) while line 10 performs operations of type (ii). Operations of type (iii) are not required. The cost of Algorithm 1 is $2nmr+O(nm)$ scalar field operations of type $\{+,-,\times,/\}$ from K, where r is the rank of A.

Many linear algebra problems involving an input matrix A can be solved by transforming A to echelon form. In particular, the *rank* of A is the number of nonzero rows in the echelon form. More importantly, the *column rank profile* of A is given by the list of column indices of the pivots, computed to be $[j_1, j_2, \ldots, j_r]$ by Algorithm 1. For example, the column rank profile of the example row echelon form in (1.1) is $[1,3,4,7,9]$. Similarly, the row rank profile is revealed to be $[1,3,4,5,6]$ by the column echelon form in (1.2). The rank profile is important information in exact linear algebra computations, and is needed in applications such as Gröbner basis computations [10] and computational number theory [22].

Transformation to echelon form can also be used for *linear solving*: given a right-hand-side vector $b \in \mathsf{K}^{m\times 1}$ in addition to $A \in \mathsf{K}^{n\times m}$, find a particular solution x such that $Ax = b$, if one exists. First, transform the augmented system $\left[\begin{array}{c|c} A & b \end{array}\right]$ to echelon form. The linear system $Ax = b$ is *consistent* if the rank of $\left[\begin{array}{c|c} A & b \end{array}\right]$ is equal to the rank of A, and *inconsistent* if the rank of $\left[\begin{array}{c|c} A & b \end{array}\right]$ is greater than the rank of A. If the system is inconsistent, a so-called *certificate of inconsistency* [11] — a row vector u such that $uA = 0$ and $ub \neq 0$ — can be read off from row $r+1$ of the transformation matrix $U \in \mathsf{K}^{n\times n}$ such

Algorithm 1 `GaussianElimination`(A)

Require: $A \in \mathsf{K}^{n \times m}$
Ensure: A is transformed to row echelon form in-place

```
1:  r := 0;
2:  for k := 1 to m do
3:    if there exists a p ∈ {r + 1, r + 2, ..., n} such that A[p, k] ≠ 0 then
4:      r := r + 1;
5:      j_r := k;
6:      if p > r then
7:        Interchange rows p and r;
8:      end if
9:      for i := r + 1 to n do
10:       Add A[i, k]/A[k, k] times row r to row i;
11:     end for
12:   end if
13: end for
```

that UA is in row echelon form. For example, using the same input matrix $A \in \mathsf{K}^{6\times 9}$ from above, the row echelon form, together with the transformation matrix, for an inconsistent system $Ax = b$ can be found by transforming the augmented matrix $\left[\begin{array}{c|c|c} A & b & I_6 \end{array}\right]$ to row echelon form.

$$\left[\begin{array}{c|c|c} A & b & I_6 \end{array}\right] \longrightarrow \left[\begin{array}{ccccccccc||c|cccccc} \circledast & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\ & & \circledast & * & * & * & * & * & * & * & * & * & * & * & * & * \\ & & & \circledast & * & * & * & * & * & * & * & * & * & * & * & * \\ & & & & & & \circledast & * & * & * & * & * & * & * & * & * \\ & & & & & & & & \circledast & * & * & * & * & * & * & * \\ & & & & & & & & & \circledast & \boxed{*} & \boxed{*} & \boxed{*} & \boxed{*} & \boxed{*} & \boxed{*} \end{array}\right]$$

Note that the trailing 6×6 matrix in the echelon form is the transformation matrix U. A certificate of inconsistency vector u is indicated by $\boxed{*}$.

If the system is consistent, a particular solution can be obtained by performing some further row operations that transform $\left[\begin{array}{c|c} A & b \end{array}\right]$ to *reduced row echelon form*, where all pivot entries are 1 and entries above the pivot entries are all zero. For example, if $b \in \mathsf{K}^{6\times 1}$ is a right hand side such that the system $Ax = b$ is consistent, the reduced row echelon

form of the augmented system $\left[\begin{array}{c|c} A & b \end{array}\right]$ for the same example above is

$$\left[\begin{array}{cccccccc|c} 1 & * & & & * & * & & * & & x_1 \\ & & 1 & & * & * & & * & & x_3 \\ & & & 1 & * & * & & * & & x_4 \\ & & & & & & 1 & * & & x_7 \\ & & & & & & & & 1 & x_9 \\ & & & & & & & & & \end{array}\right]$$

A particular solution $x \in \mathsf{K}^{9\times 1}$ to $Ax = b$ can be read off from the last column: set x_1, x_3, x_4, x_7, x_9 to be entries $1, 3, 4, 7, 9$ of x, and set the other entries of x to be zero.

We can define these two problems, which are the focus of this thesis, as follows.

- LinSys: Given a $b \in \mathsf{K}^{n\times 1}$, compute a particular solution $x \in \mathsf{K}^{m\times 1}$ to $Ax = b$, or a certificate of inconsistency [11]: a row vector $u \in \mathsf{K}^{1\times n}$ such that $uA = 0$ and $ub \neq 0$.
- RankProfiles: Compute the rank r together with the lexicographically minimal lists $[i_1, i_2, \ldots, i_r]$ and $[j_1, j_2, \ldots, j_r]$ of row and column indices of A such that these rows and columns of A, respectively, are linearly independent.

Although transformation to echelon form will solve these problems, it is not necessarily required. Much research has been devoted to obtaining algorithms for these problems which have improved asymptotic running time compared to that of Algorithm 1. Before summarizing previous results in the next section, we mention three main directions that research has taken.

D1 Improved cost estimates for rectangular and low rank systems.
Input matrices may be rectangular and may be rank deficient and it would be desirable for an algorithm to exploit this. As mentioned above, classical Gaussian elimination already gives an algorithm with a *rank sensitive* cost estimate $2nmr + O(nm)$, where r is the rank, which improves on the coarse, single-parameter cost estimate $\max(n, m)^3$ whenever $n < m$, $m < n$, or $r < \min(n, m)$. Another goal is to decouple the cubic part of the complexity from the matrix dimensions n and m to achieve algorithms with running time on the order of, for example, $O(r^3 + nm)$ instead of $O(nmr)$.

D2 Exploit possible sparsity of the input matrix.
Input matrices may be sparse, that is, if we let $|A|$ denotes the number of nonzero

entries in A, then we may have $|A| \in o(nm)$. Sparse matrices over finite fields arise in applications such as factoring integers [17] and Gröbner basis computations where huge sparse linear systems must be solved [9]. Unfortunately, Gaussian elimination tends to quickly cause the matrix being transformed to fill in and may not give improved worst case time bound in terms of $|A|$. Many heuristics have been designed to reduce fill-in during Gaussian elimination (see Section 2.1 of [7] for a discussion) but they do not in general guarantee improved worst case running time bounds, that is, the worst case cost when $|A| \in o(nm)$ is not better than when $|A| = \Theta(nm)$. Projection methods, also called or black-box methods [16], avoid fill in by only using A to perform matrix×vector products and for sparse matrices and can have improved running time compared to elimination methods.

D3 Exploit fast matrix multiplication.
Let ω, $2 < \omega \leq 3$, be the exponent of matrix multiplication. The best currently known upper bound for ω is 2.3727 [29]. Block recursive versions of Gaussian elimination [15, 24, 25] can compute echelon forms in time $O(nmr^{\omega-2})$ field operations from K. Fast matrix multiplication is highly important from a practical, implementation point of view because it exploits cache effects. High-performance, exceedingly optimized matrix multiplication implementations such as ATLAS BLAS [27] and GotoBLAS [13] are used to obtain fast implementations of linear algebra algorithms for matrices over finite fields [3, 5].

In this thesis we give randomized algorithms for problems LinSys and RankProfiles that support the currently fastest running times in terms of the parameters n, m, r and $|A|$ for many cases of these input parameters. The algorithms we give are applicable to dense and spare systems, and naturally take advantage of sparsity when it is present. Our algorithm for RankProfile can exploit matrix multiplication.

We first summarize previous complexity results and then discuss our new algorithms and cost estimates for solving the above problems. Throughout, running times of algorithms are given in terms of the number of required field operations from K.

1.1 Previous work

A lot of effort has been devoted to designing and analyzing algorithms for problems LinSys and RankProfiles that have a running time that is sensitive to the rank r of A. We have seen that classical Gaussian elimination already solves both problems in time $O(nmr)$.

Using block recursive methods [6, 15] the running time is reduced to $O(nmr^{\omega-2})$ where ω is the exponent of matrix multiplication. A somewhat faster algorithm is possible when A is sparse [30]. Problem LinSys can be solved in time $O((n+m)r^2)$ using a so-called oracle based variant [19, Section 2] of Gaussian elimination that examines at most $r+1$ rows and r columns of A. Note that if $r^2 < \min(m,n)$, then $(n+m)r^2 < nm$, so the running time of the oracle solver may be sublinear in the size of the input.

Now consider randomized algorithms. If only the rank of A is required, then efficient preconditioners [2, 16] give a Monte Carlo algorithm with running time

$$(r^\omega + nm)^{1+o(1)}. \tag{1.3}$$

Here, the $o(1)$ in the exponent of cost estimates captures some missing $\log n$ and $\log m$ factors. The running time bound (1.3) improves on Gaussian elimination by not only incorporating matrix multiplication but also decoupling the "matrix multiplication" component from the row and column dimension n and m. However, the method does not exploit any possible sparsity of A.

Heuristics and algorithms (with implementations) for computing the rank of very large (out of core) dense matrices modulo a prime p have been developed [18, 20]. Now let $\mu(A)$ denote the time required to multiply a vector by A. For a sparse or structured matrix we may have $\mu(A) \in o(nm)$. In particular, if $|A|$ denotes the number of nonzero entries in A then we can take $\mu(A) \in O(|A|)$. Black-box approaches [16, 28] combined with early termination [8] can compute r in time $(\mu(A)\, r)^{1+o(1)}$; incorporating early termination into the black-box algorithm in [11] would seem to give a Las Vegas algorithm for LinSys with the same running time. In terms of the rank, none of the randomized approaches just mentioned recover the rank profile, or even determine a set of r linearly independent columns of A.

Our work in this thesis is motivated by a recent breakthrough by Cheung, Kwok and Lau [4], who give an algorithm for computing the rank of A in time $(r^\omega + n + m + |A|)^{1+o(1)}$. Note that the $|A|$ term in the cost estimate is optimal, since changing a single entry of A might modify the rank. They also show how to compute a set of r linearly independent columns of A in the same time. However, the columns computed may not correspond to the column rank profile. For an extensive list of applications (with citations) of fast rank computation to combinatorial optimization and exact linear algebra problems we refer to [4].

1.2 New algorithms

In this thesis we give new randomized algorithms for the problems LinSys and RankProfiles that have improved running times in the special case when the rank r is small compared to the row dimension n or column dimension m, or both. We give Monte Carlo randomized algorithms for RankProfiles that support the running time bounds $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$ and $(r^\omega + n + m + |A|)^{1+o(1)}$, depending on whether or not matrix multiplication is incorporated. We give a Las Vegas randomized algorithm for LinSys that has running time $2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)}$, where $|R|$ and $|C|$ are the number of nonzero entries in the subset of at most $r + 1$ rows and r columns of A, respectively, that are examined by the solver: at least $(m - r)(n - r - 1)$ entries of A are not examined. The algorithm of Cheung, Kwok and Lau [4] also gives a solution to problem LinSys in time $(r^\omega + |A|)^{1+o(1)}$. The algorithm we give here for LinSys has the advantage that we examine at most $r + 1$ rows and r columns of A; in particular, we may have $|R| + |C| < |A|$.

Now consider the new cost estimates compared to the black box approaches mentioned above. The black box approaches can find the rank in a Monte Carlo fashion and solve LinSys in time $(|A|\, r)^{1+o(1)}$. Thus, the algorithms we present here are asymptotically faster whenever $r^2 < |A|$. In terms of space, the black box approaches use space for only $O(n+m)$ additional field elements, while our algorithms use $O(r^2)$ additional space. Thus, if $r^2 \in \omega(n+m)$ then the black box approach requires asymptotically less space. However, the black box approach can only compute the rank, but does not recover the rank profile, or even determine a set of r linearly independent columns of A.

In later chapters we develop our algorithm for LinSys and RankProfiles directly for the general case of an input matrix $A \in \mathsf{K}^{n\times m}$ of (unknown) rank r. But to help explain our main technical contributions, we first consider the special case of an input matrix with full row rank, and then with full column rank.

Let $A \in \mathsf{K}^{n\times m}$ be an input matrix that has full row rank n, and consider the problem of computing the column rank profile of A. Gaussian elimination on A uses $O(n^2m)$ time to compute a lower triangular matrix L such that, up to a row permutation, the matrix LA is upper triangular. The first few rows of such a transformation are shown in Figure 1.1. The columns of A corresponding to the pivot entries of LA (the first nonzero zero entry in each row) are thus linearly independent. But even if L is given, computing LA explicitly uses $O(n^2m)$ time using standard matrix multiplication. Instead of computing LA explicitly, we show how to construct from the columns of A in time $|A|^{1+o(1)}$ a randomized data structure — a *linear independence oracle* — that allows one to apply binary search to find

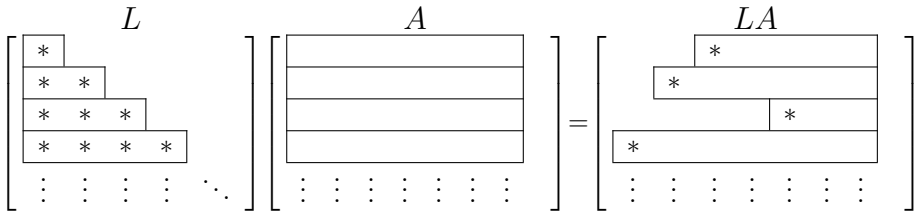


Figure 1.1: Example of pivot locations

the pivot locations in LA in a Monte Carlo fashion in time $(n^2 \log m)^{1+o(1)}$. This gives a Monte Carlo algorithm to find the column rank profile of a full row rank $A \in \mathsf{K}^{n \times m}$ in time $2n^3 + (n^2 + m + |A|)^{1+o(1)}$.

Now let $A \in \mathsf{K}^{n \times m}$ have full column rank m, and consider the problem of computing the row rank profile of A. Of course, we could simply apply the algorithm for column rank profile to the transpose of A. But with an eye to the general case (A with unknown rank) we develop an alternative method based on incorporating a linear independence oracle for the rows of A into the oracle linear solving algorithm [19]. We prove that the row rank profile of a full column rank $A \in \mathsf{K}^{n \times m}$ can be computed in a Monte Carlo fashion in time $2m^3 + (m^2 + n + |A|)^{1+o(1)}$ by running the linear solver algorithm with a right-hand-side vector b that is uniformly and randomly sampled from the column space of A.

Incorporating linear independence oracles for both the rows and columns of A, we achieve the running time bound $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$ for RankProfiles. The leading term $2r^3$ in the cost estimate arises from our use of an iterative algorithm to compute, for $s = 1, 2, \ldots, r$, the inverse of the leading principal $s \times s$ submatrix B_s of an $r \times r$ matrix B that has generic rank profile, and whose rows are given from first to last, one at a time, for $s = 1, 2, \ldots, r$. These inverses are used to compute a sequence of subsystem solutions $B_s^{-1} b_s$ for $s = 1, 2, \ldots, r$, where $b_s \in \mathsf{K}^{s \times 1}$ is the leading subvector of a $b \in \mathsf{K}^{r \times 1}$, as shown by the following augmented system.

$$\left[\begin{array}{c|c||c} B_s & * & b_s \\ \hline * & * & * \end{array}\right] = \left[\begin{array}{c|c||c} B & * & b \end{array}\right]$$

This motivates the definition of the following problem.

- OnlineSystem: Let $B \in \mathsf{K}^{r \times r}$ with generic rank profile and $b \in \mathsf{K}^{r \times 1}$ be given. Suppose the rows of the augmented system $\left[\begin{array}{c|c} A & b \end{array}\right]$ are given one at a time, from first to last. As soon as rows $1, 2, \ldots, s$ of $\left[\begin{array}{c|c} B & b \end{array}\right]$ are given, produce the subsystem solution $B_s^{-1} b_s$, for $s = 1, 2, \ldots, r$.

In Chapter 8 we show how to compute the sequence $B_1^{-1}b_1$, $B_2^{-1}b_2, \ldots, B_r^{-1}b_r$ in an online fashion in time $O(r^\omega)$, effectively allowing matrix multiplication to be incorporated into our rank profile algorithm. Together with a Toeplitz preconditioner [16], we can compute the row rank profile of a full column rank matrix A in time $(r^\omega + n + m + |A|)^{1+o(1)}$. Combined with the algorithm supporting [4, Theorem 2.11] for computing a maximal rank subset of of linearly independent columns, this gives a Monte Carlo algorithm for RankProfiles in time $(r^\omega + n + m + |A|)^{1+o(1)}$.

To summarize, the rest of this thesis is organised as follows. In Chapter 2 we recall the deterministic oracle linear solving algorithm [19]. Next, Chapters 3–8 give our novel technical contributions, which can be partitioned into four main contributions **C1**–**4** as follows.

C1
- Chapter 3: We show how the deterministic oracle linear solver [19, Section 2] can be used to simultaneously recover the row and column rank profiles of an input matrix A in a Monte Carlo fashion by giving as input to the solver a system $\left[\begin{array}{c\|c} A & b \end{array}\right]$ where b is uniformly and randomly sampled form the column space of A.

C2
- Chapter 4: We design and analysis a new online randomized data structure — *the linear independence oracle* — that allows applying binary search for the detection of linearly independent rows and columns of a matrix A.

C3
- Chapter 5: We show how to incorporate linear independence oracles for both the rows and columns of A into the oracle-based linear solving algorithm to obtain a randomized Las Vegas algorithm for problem LinSys that has running time $2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)}$.
- Chapter 6: We show how to incorporate linear independence oracles into the rank algorithm of Chapter 3 to obtain a randomized Monte Carlo algorithm for problem RankProfiles that has running time $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$.

C4
- Chapter 7: We design a relaxed representation of the inverse of a matrix, and design and analyse an online algorithm that gives a solution to problem OnlineInverse that supports the running time bound $O(r^\omega)$.
- Chapter 8: We establish that problem RankProfiles can be solved in a Monte Carlo fashion in time $(r^\omega + n + m + |A|)^{1+o(1)}$.

Finally, Chapter 9 gives a conclusion along with topics for future research.

Contributions **C1**–**4** have been published [23]. While the full exposition of **C4** appears for the first time in this thesis, a poster presentation of the main ideas has appeared [26].

Throughout the paper we use the following notation. For a list $\mathcal{P}$ of distinct row indices and $\mathcal{Q}$ of distinct column indices, we write $A^{\mathcal{P}}$ to denote the submatrix of A consisting of rows $\mathcal{P}$, $A_{\mathcal{Q}}$ to denote the submatrix consisting of columns $\mathcal{Q}$, and $A^{\mathcal{P}}_{\mathcal{Q}}$ to denote the submatrix consisting of the intersection of rows $\mathcal{P}$ and columns $\mathcal{Q}$ of A.

Chapter 2

Oracle linear system solving

Let $A \in \mathsf{K}^{n\times m}$ and $b \in \mathsf{K}^{n\times 1}$ be given. Oracle-based elimination [19] is a variation of Gaussian elimination that either finds a particular solution to the linear system $Ax = b$ or proves that the system is inconsistent. The cost of the algorithm is $O((n+m)r^2)$ field operations from K, where r is the rank of A. This cost estimate, which can be $o(nm)$ for small values of r, can be achieved because the algorithm never examines more than r columns and $r+1$ rows of the input matrix. The name was chosen because the target vector b is used as an oracle to determine a set of linearly independent rows of the matrix A.

The oracle based solver proceeds in stages for $s = 0, 1, \ldots$. The goal at stage s is to determine a list of row indices $\mathcal{P} = [i_1, i_2, \ldots, i_s]$ and column indices $\mathcal{Q} = [j_1, j_2, \ldots, j_s]$ such that

- $A^{\mathcal{P}} \in \mathsf{K}^{s\times m}$ has full row rank s,
- $A_{\mathcal{Q}} \in \mathsf{K}^{n\times s}$ has full column rank s, and
- $A^{\mathcal{P}}_{\mathcal{Q}} \in \mathsf{K}^{s\times s}$ is nonsingular.

To start the process for stage $s = 0$ the lists $\mathcal{P}$ and $\mathcal{Q}$ are simply initialized to be empty. To complete a stage s for some $s > 0$, the algorithm takes the following two steps.

1. If
$$b - A_{\mathcal{Q}} \left(A^{\mathcal{P}}_{\mathcal{Q}}\right)^{-1} b^{\mathcal{P}} \tag{2.1}$$

is the zero vector, then construct a particular solution to $Ax = b$ by setting x to be the zero vector except with entries at index $j_1, \ldots, j_{s-1}$ equal to entries at index $1, \ldots, s-1$ of $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} b^{\mathcal{P}}$. Otherwise, let i_s be the index of the first nonzero entry of the column vector in (2.1) and go to step 2.

2. If
$$A^{[i_s]} - A_{\mathcal{Q}}^{[i_s]} (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} A^{\mathcal{P}} \tag{2.2}$$
is the zero vector, then the system is inconsistent. A so-called certificate of inconsistency [11] can be constructed by setting $u \in \mathsf{K}^{1\times n}$ to be the zero vector, except with entry at index $i_1, \ldots, i_{s-1}$ equal to the entry at index $1, 2, \ldots, s-1$ of $-A_{\mathcal{Q}}^{[i_s]} (A_{\mathcal{Q}}^{\mathcal{P}})^{-1}$, and entry at index i_s equals to 1. The vector u has the property that $uA = 0$ and $ub \neq 0$. Otherwise, set j_s to be the index of the first nonzero entry of the row vector in (2.2), update $\mathcal{P} = [i_1, i_2, \ldots, i_s]$ and $\mathcal{Q} = [j_1, j_2, \ldots, j_s]$, and proceed to stage $s+1$.

Note that for $s = 1$, (2.1) is simply equal to the b vector. Also note that $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1}$ need not be recomputed from scratch at each stage. Rather, the inverse for the next stage can be computed in $O(s^2)$ field operations from the inverse of the current stage by performing a rank 1 update (Lemma 2). Since s is bounded by $r+1$, all the inverses required can be computed in time $O(r^3)$. Similarly, computing the vectors in (2.1) and (2.2) costs $O(sn)$ and $O(sm)$ time, respectively, for an overall cost of $O((n+m)r^2)$.

Theorem 1. *There exists a deterministic algorithm* `OracleSolve`(A, b) *that takes as input* $A \in \mathsf{K}^{n\times m}$ *and* $b \in \mathsf{K}^{n\times 1}$*, and returns as output either "consistent,* x*,* $\mathcal{P}$*,* $\mathcal{Q}$*" or "inconsistent,* u*,* $\mathcal{P}$*,* $\mathcal{Q}$*" as described above. The cost of the algorithm is* $O((n+m)r^2)$ *field operations from* K*.*

Although the overall cost estimate $O(r^3)$ for computing the required inverses does not dominate the running time of algorithm `OracleSolve`, it might dominate the running time for the improved variations of algorithm `OracleSolve` that we give in later sections. Thus, it will be useful here to derive the leading constant in this asymptotic bound for updating the inverses. Let $\mathcal{P} = [i_1, i_2, \ldots, i_{s-1}]$ and $\mathcal{Q} = [j_1, j_2, \ldots, j_{s-1}]$ be as at the start of stage s, and let i_s and j_s be as computed in steps 1 and 2. If we set $u = A_{[j_s]}^{\mathcal{P}} \in \mathsf{K}^{(s-1)\times 1}$, $v = A_{\mathcal{Q}}^{[i_s]} \in \mathsf{K}^{1\times(s-1)}$ and $d = A_{[j_s]}^{[i_s]} \in \mathsf{K}^{1\times 1}$, then at stage $s+1$ we need the inverse of
$$\left[\begin{array}{c|c} A_{\mathcal{Q}}^{\mathcal{P}} & u \\ \hline v & d \end{array}\right] \in \mathsf{K}^{s\times s}. \tag{2.3}$$

Lemma 2. *If the inverse* $B := (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \in \mathsf{K}^{(s-1)\times(s-1)}$ *is precomputed, then the inverse of the matrix in (2.3), if it exists, is equal to*

$$\left[\begin{array}{c|c} B + (Bu)w(vB) & -(Bu)w \\ \hline -w(vB) & w \end{array}\right], \tag{2.4}$$

with $w = (d - v(Bu))^{-1}$*, and can be computed in* $6s^2 + O(s)$ *field operations from* K*.*

Proof. Correctness of the inverse is verified by direct computation. The total cost of the matrix×vector product Bu, the vector×matrix product vB, the outer product $(Bu)(vB)$, and the addition $B + (Bu)w(vB)$, is $6s^2 + O(s)$ field operations from K. The remaining dot products and scalar operations have cost $O(s)$. □

Since s is bounded by $r + 1$, all the inverses required can be be computed in time $\sum_{s=1}^{r+1}(6s^2 + O(s)) = 2r^3 + O(r^2)$.

Remark 3. *The vector×matrix product* $A_{\mathcal{Q}}^{[i_s]}\,(A_{\mathcal{Q}}^{\mathcal{P}})^{-1}$ *in (2.2) for stage* s *is exactly the vector×matrix product* vB *used to update the inverse for stage* $s+1$ *in the proof of Lemma 2.*

Remark 4. *Assuming that* Bu *and* vB *in (2.4) have been precomputed, the matrix×vector product* $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1}\,b^{\mathcal{P}}$ *in (2.1) for stage* $s + 1$ *can be computed in* $O(s)$ *field operations by applying the formula for* $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1}$ *shown in (2.4) to* $b^{\mathcal{P}}$*, using the fact that* $Bb^{[i_1,\ldots,i_{s-1}]}$ *is known from the previous stage.*

Chapter 3

Randomized rank profiles

In this section we establish that if b is uniformly and randomly sampled from the column space of A, then the list of row indices $\mathcal{P}$ and the list of column indices $\mathcal{Q}$ returned by $\texttt{OracleSolve}(A, b)$ will be the row and column rank profiles of A with high probability. (The list $\mathcal{Q}$ may need to be sorted in increasing order.) Algorithm 2 outlines this randomized algorithm for RankProfiles.

Algorithm 2 $\texttt{RandomRankProfile}(A)$

Require: $A \in \mathsf{K}^{n \times m}$
Ensure: $s \in \mathbb{Z}_{\geq 0}$, $\mathcal{P}$, $\mathcal{C}$
1: Choose a $w \in \mathsf{K}^{m \times 1}$ uniformly and randomly.
2: $b := Aw$
3: $*, *, \mathcal{P}, \mathcal{Q} := \texttt{OracleSolve}(A, b)$
4: $\mathcal{C} := \text{sort}(\mathcal{Q})$
5: Return length($\mathcal{P}$), $\mathcal{P}$, $\mathcal{C}$

Lemma 5. *Let s, $\mathcal{P}$ and $\mathcal{C}$ be the output of Algorithm 2* $\texttt{RandomRankProfile}$ *for an input matrix A of rank r. If $s = r$ then $\mathcal{C}$ is the column rank profile of A.*

Proof. If $s = r$ then the row space of $A^{\mathcal{P}}$ is equal to the row space of A. Now note that $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} A^{\mathcal{P}}$ is equal to a row permutation of the reduced row echelon form of A. Since the list $\mathcal{Q}$ corresponds to the column indices of the pivot entries in $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1} A^{\mathcal{P}}$ it is, up to sorting, equal to the column rank profile of A. □

Theorem 6. *The output $\mathcal{P}$ and $\mathcal{C}$ of Algorithm 2* `RandomRankProfile` *is the row and column rank profile of A with probability at least $(1 - 1/\#\mathsf{K})^r$, where r is the rank of A.*

The following two lemmas contribute to the proof of Theorem 6.

Lemma 7. *Let $A \in \mathsf{K}^{n\times m}$ be a matrix of rank r and let $b \in \mathsf{K}^{n\times 1}$ be in the column space of A. Let*

- *$\bar{A} \in \mathsf{K}^{r\times m}$ be the submatrix of A comprised of the rank profile rows and let $\bar{\mathcal{P}}$ be the list of row indices returned by* `OracleSolve`$(\bar{A}, b)$,
- *$\mathcal{P}$ be the row indices returned by* `OracleSolve`(A, b).

If $\bar{\mathcal{P}} = [1, 2, ..., r]$, then $\mathcal{P}$ is the row rank profile of A.

Proof. Assume $\bar{\mathcal{P}} = [1, 2, \ldots, r]$. Then the only difference in the computations done by the call `OracleSolve`(A, b) compared to `OracleSolve`$(\bar{A}, b)$ occurs in step 1 when finding the first nonzero entry in the vector in (2.1): precisely the indices corresponding to rows not belonging to the rank profile will be skipped over, since these rows of the augmented system $\left[\begin{array}{c\|c} A & b \end{array}\right]$ are linearly dependent on the previous rows. □

Lemma 8. *Let K be a finite field. If $A \in \mathsf{K}^{n\times m}$ has full row rank n, and $w \in \mathsf{K}^{m\times 1}$ is chosen uniformly and randomly, then Aw is chosen uniformly and randomly from $\mathsf{K}^{n\times 1}$.*

Proof. Corresponding to $A \in \mathsf{K}^{n\times m}$ there always exists a nonsingular matrix $U \in \mathsf{K}^{m\times m}$ such that $AU = \left[\begin{array}{c|c} I_n & 0 \end{array}\right]$. Then $Aw = AU(U^{-1}w) = b$. When $w \in \mathsf{K}^{m\times 1}$ is uniformly and randomly chosen from K, $U^{-1}w$ is also uniformly and randomly chosen from $\mathsf{K}^{m\times 1}$. The result now follows by noting that b is equal to the first n entries of $U^{-1}w$. □

Proof of Theorem 6. By Lemma 7, it will be sufficient to prove the theorem with input matrix $A \in \mathsf{K}^{r\times m}$ of full row rank r. When w is chosen uniformly and randomly from $\mathsf{K}^{m\times 1}$, vector $b = Aw \in \mathsf{K}^{r\times 1}$ is also chosen uniformly and randomly from $\mathsf{K}^{r\times 1}$ (Lemma 8). Since A is full rank, $\mathcal{P}$ is the row rank profile of A if $i_s = s$ for $1 \le s \le r$.

For some $s \ge 1$, suppose $\mathcal{P} = [i_1, i_2, ..., i_{s-1}]$ at the start of stage s has been computed correctly to be $[1, 2, \ldots, s-1]$. Then we claim that at stage s the probability that step 1 of the algorithm computes $i_s = s$ is equal to $1 - 1/\#\mathsf{K}$. To see this, note that the entry of (2.1) at index s is equal to

$$b[s] - A_{\mathcal{Q}}^{[s]} \left(A_{\mathcal{Q}}^{\mathcal{P}}\right)^{-1} b^{\mathcal{P}}.$$

Since $b[s]$ is chosen uniformly and randomly from field K, the probability that $b[s] = A_{\mathcal{Q}}^{[s]} \left(A_{\mathcal{Q}}^{\mathcal{P}}\right)^{-1} b^{\mathcal{P}}$ is bounded by $1/\#\mathsf{K}$.

Since the entries in b are chosen independently, the success probability at each stage can be multiplied together to obtain $(1 - 1/\#\mathsf{K})^r$ for the probability that the algorithm returns $\mathcal{P} = [1, 2, \ldots, r]$.

Finally, if $\mathcal{P}$ is the rank profile of A, then by Lemma 5, $\mathcal{C}$ will be the column rank profile also. □

For Algorithm 2, we remark that the output can satisfy $\text{length}(\mathcal{P}) = \text{rank}(A)$ even if $\mathcal{P}$ is not the rank profile of A. Consider the following example over $\mathsf{K} = \mathbb{Z}_3$:

$$A = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}, w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, b = Aw = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

At stage $s = 1$, the vector (2.1) in step 1 is equal to b. The first entry in (2.1) is zero, so $i_1 = 2$. Therefore, `OracleSolve`(A, b) will return $\mathcal{P} = [2, 1]$ for the row rank profile, which is incorrect.

Chapter 4

Linear independence oracles

Throughout this section we fix $R \in \mathsf{K}^{r \times m}$ and let $v \in \mathsf{K}^{1 \times r}$ be given. We describe the construction of a randomized binary tree data structure T that will allow us to either determine if v is in the left nullspace of R, that is, if vR is the zero vector, or to find the first nonzero entry of $vR \in \mathsf{K}^{1 \times m}$, in a Monte Carlo fashion in time $O(r \log m)$. We call T a *linear independence oracle* for R. Constructing a single linear independence oracle T for R costs $O(mr)$ field operations, which would seem to preclude its usefulness since we can compute $vR \in \mathsf{K}^{1 \times m}$ explicitly in the same time. However, our algorithm for constructing T is online. We will show how to construct in overall $O(mr)$ time a sequence of linear independence oracles T_s for the submatrices $R^{[1,2,\ldots,s]}$ for R, $s = 1, 2, \ldots, r$.

4.1 Definition

By augmenting R with at most $m-2$ zero columns we may assume without loss of generality that m is a power of two. A linear independence oracle T for R, shown in Figure 4.1, is a perfect binary tree of height $h := \log_2 m$, that is, with m leaf nodes. The nodes of the last level of the tree, from left to right, are associated with (i.e., store) the corresponding columns $R[1], R[2], \ldots, R[m]$ of R. In addition to R, the definition of the tree is based on a given list of elements $\alpha_1, \alpha_2, \ldots, \alpha_{m-1} \in \mathsf{K}$, which we associate with the internal nodes of T according to level order traversal. In a bottom-up fashion, for levels $h-1, h-2, \ldots, 0$, associate with each internal node the vector obtained by adding the vector associated to the left child with α_* times the vector associated with the right child. For example, for the left child of the root we have $R_{1 \sim \frac{m}{2}} = R_{1 \sim \frac{m}{4}} + \alpha_2 R_{\frac{m}{4}+1 \sim \frac{m}{2}}$. Note that, by construction, $R_{a \sim b}$ is a vector that is a linear combination of $R[a], R[a+1], \ldots, R[b]$. We call T a *linear*

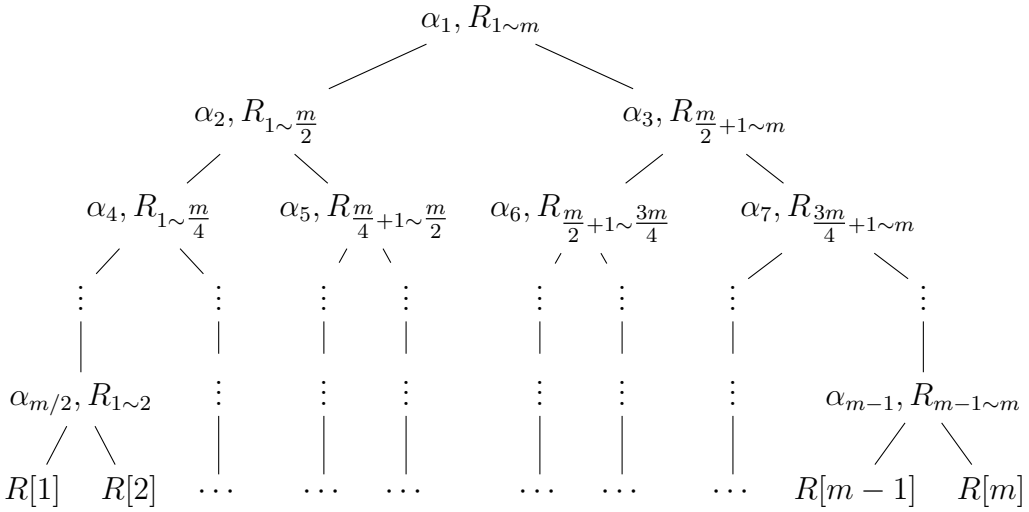


Figure 4.1: Oracle tree

independence oracle for R based on the parameters $\alpha_1, \ldots, \alpha_{m-1}$. In what follows it will be helpful to abuse notation somewhat and simply identify the nodes of T with the vectors stored at the node. For example, v is non-orthogonal with a node x if the dot product of v with x is nonzero.

4.2 Output

On the one hand, if v is in the left nullspace of R, then v is orthogonal with the root $R_{1\sim m}$ as well, since by construction $R_{1\sim m}$ is a linear combination of the columns of R. On the other hand, if $vR_{1\sim m}$ is nonzero, then because each internal node of T is a linear combination of its children, there must be a path from the root down to a leaf with v non-orthogonal to all nodes on the path.

Definition 9. *The* output of a linear independence oracle T with respect to a $v \in \mathsf{K}^{1\times r}$ *is either*

- *"v is in the left nullspace of R" if $vR_{1\sim m}$ is zero, or,*
- *the minimal j for which v is non-orthogonal with all nodes on the path from $R[j]$ back up to the root.*

Lemma 10. *The output of T with respect to v can be found in time $O(r \log_2 m)$.*

Proof. If $vR_{1\sim m}$ is zero we are done. Otherwise, use a straightforward recursive search starting at the root node. For example, if $vR_{1\sim \frac{m}{2}} \neq 0$ then recurse on the left subtree. If $vR_{1\sim m/2} = 0$ then necessarily $vR_{\frac{m}{2}+1\sim m} \neq 0$, so recurse on the right subtree. After arriving at a leaf node $R[j]$ report j. The cost is h dot products of length r. □

Definition 11. *T is* correct with respect to *a $v \in \mathsf{K}^{1\times r}$ if either v is in the left nullspace of R, or the output of T with respect to v is the minimal j such that v is non-orthogonal to $R[j]$.*

Correctness of T will depend on the choice of the parameters $\alpha_1, \ldots, \alpha_{m-1}$ used during the construction. We first deal with construction of T before considering the probability of correctness.

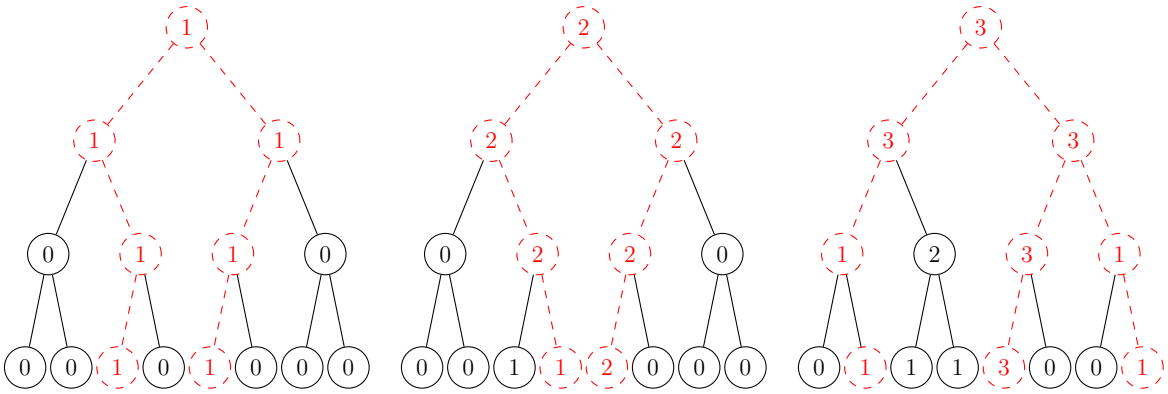
4.3 Construction

Theorem 12. *A sequence of linear independence oracles T_s for $R^{[1,2,\ldots,s]} \in \mathsf{K}^{s\times m}$, $s = 0, 1, \ldots, r$, all based on the parameters $\alpha_1, \ldots, \alpha_{m-1}$, can be constructed in total time $O(\min(mr, m + |R| \log_2 m))$, where $|R|$ denotes the number of nonzero entries of R.*

Proof. We will construct T_s for $s = 0, 1, \ldots, r$ in succession. For efficiency, each column vector associated with a node in tree T_s should be represented as a singly linked list of nonzero elements, that is, each node in the list contains the index and the nonzero element of the vector stored at that index. Initialize T_0 to be a perfect binary tree with each node associated with a vector of length zero. The cost of this initialization is accounted for by the "$m+$" term in the cost estimate.

Given T_{s-1} for some $s > 0$, the linear oracle T_s is constructed from T_{s-1} and the next row vector $R^{[s]}$ in a bottom up fashion: augment the column vectors associated to the nodes in level i of T_{s-1} for $i = h, h-1, \ldots, 1$. Since there are $2m-1$ nodes in the tree the cost of obtaining T_s from T_{s-1} is bounded by $O(m)$. Since $1 \leq s \leq r$ the total cost is $O(mr)$.

Note that only the leaf nodes associated to nonzero elements of row $R^{[s]}$ may need to have their associated vectors modified. Similarly, internal nodes of the tree may need to be modified only if they had a child node that was modified. The cost of modifying each level is thus $|R^{[s]}|$. (See Example 13 below.) Since there are $1 + \log_2 m$ levels, the total cost



(a) T_1 (b) T_2 (c) T_3

Figure 4.2: T_1, T_2, and T_3

of constructing T_s from T_{s-1} is $O(|R^{[s]}| \log m)$. Summing this estimate for $s = 1, 2, \ldots, r$, that is, over all rows of R, gives the claimed result. □

Example 13. *Suppose* $r = 3$, $m = 8$ *and*

$$R = \begin{array}{|cccccccc|} \multicolumn{8}{c}{\textit{1}\quad\textit{2}\quad\textit{3}\quad\textit{4}\quad\textit{5}\quad\textit{6}\quad\textit{7}\quad\textit{8}} \\ \hline & & * & & * & & & \\ \hline & & & * & * & & & \\ \hline & * & & & * & & & * \\ \hline \end{array} \in \mathsf{K}^{3\times 8},$$

with zero entries left blank and nonzero entries indicated with a $*$*. All nodes in the initial tree* T_0 *store empty linked lists. The construction of* T_1, T_2 *and* T_3 *are shown in Figure 4.2. For brevity, for each node in the tree only the length of the list, corresponding to the number of nonzero entries in the vector, is indicated. The modified nodes (and their paths) are highlighted in dashed red.*

4.4 Probability of correctness

Lemma 14. *Let* K *be a finite field and* $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$ *be chosen uniformly and randomly. If* T *is a linear independence oracle for* $R \in \mathsf{K}^{r\times m}$ *based on* $\alpha_1, \ldots, \alpha_{m-1}$*, then* T *is correct with respect to* v *with probability at least* $(1 - 1/\#\mathsf{K})^{\log_2 m}$*.*

Proof. If $vR = 0$, then T is correct independent of the choice of the $\alpha_1, \ldots, \alpha_{m-1}$, so assume $vR \neq 0$ and let j be the minimal index such that $vR[j] \neq 0$. Consider the path from $R[j]$ up to the root. If the sibling of $R[j]$ is $R[\bar{j}]$ then the parent of $R[j]$ is either $R[\bar{j}] + \alpha_* R[j]$ or $R[j] + \alpha_* R[\bar{j}]$, depending on whether or not j is even or odd, respectively. In either case, since $vR[j] \neq 0$ there is at most one choice of α_* such that v is orthogonal with the parent of $R[j]$. The same argument applies for the other internal nodes on the path from $R[j]$ up to the root. Since the α_* associated with each internal node are chosen uniformly and independently from K, and the number of internal nodes on the path is $h = \log_2 m$, the result follows. □

Now, instead of a single $v \in \mathsf{K}^{1\times r}$, let a sequence of vectors $v_s \in \mathsf{K}^{1\times s}$ for $s = 1, 2, \ldots, r$ be given in addition to $R \in \mathsf{K}^{r\times m}$.

Corollary 15. *Let K be a finite field and $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$ be chosen uniformly and randomly. If $(T_s)_{1\leq s\leq r}$ is a sequence of linear independence oracles for $(R^{[1,2,\ldots,s]})_{1\leq s\leq r}$, all based on the same $\alpha_1, \ldots, \alpha_{m-1} \in \mathsf{K}$, then the $(T_s)_{1\leq s\leq r}$ are correct with respect to $(v_s)_{1\leq s\leq r}$ simultaneously with probability at least $(1 - r/\#\mathsf{K})^{\log_2 m}$.*

Proof. For $1 \leq s \leq r$, let $\bar{v}_s \in \mathsf{K}^{1\times r}$ be the vector obtained from v_s by augmenting with $r - s$ zeroes. Then T_s is correct with respect to v_s precisely when T is correct with respect to $\bar{v}_s$. For all $s \in [1, 2, \ldots, r]$ such that $\bar{v}_s R \neq 0$, consider the path from the leftmost leaf node in T that is non-orthogonal to $\bar{v}$ back up to the root. There will be at most r such paths. (See Figure 4.3 for an example illustrating eight such paths.) Consider the choice of α_* for the nodes at level $i = h - 1, h - 2, \ldots, 0$ of T. In the best case, all the paths at level i from the leaf nodes back up to the root are independent, so we can multiply the probability of success at each internal node to get the estimate $(1 - 1/\#\mathsf{K})^r$ that all the α_* intersecting a path at level i are well chosen. However, as paths rise they may join implying that the same α_* has to be well chosen for more than one path. The worst case occurs near the root node when possibly all paths converge. In the worst case, we can bound from below the probability that a given α_* is good for all paths is at least $1 - r/\#\mathsf{K}$. Since there are $\log_2 m$ internal levels, and the α_* at each level are chosen independently, the result follows. □

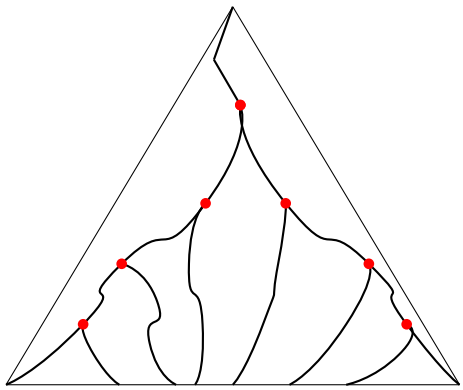


Figure 4.3: Non-orthogonal paths in a linear independence oracle

Chapter 5

Faster linear system solving

The computations that dominated the running time in the deterministic oracle solver described in Section 2 were to determine, at stage $s = 1, 2, \ldots$, the first nonzero entry in $b - A_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1}\, b^{\mathcal{P}}$ and $A^{[i_s]} - A^{[i_s]}_{\mathcal{Q}}\, (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \cdot A^{\mathcal{P}}$ in steps 1 and 2, respectively. We use "$\cdot$" to emphasis that the product is not directly computed. We can recast the computation in step 2 as follows. First compute

$$v_s := \left[\ -A^{[i_s]}_{\mathcal{Q}}\, (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} \ \middle|\ 1\ \right] \in \mathsf{K}^{1\times s}$$

in $2s^2 + O(s)$ field operations. By Remark (3), the leading term $2s^2$ in this cost estimate is already counted in the worst case cost estimate for updating all the required inverses. Now set

$$R^{[1,2,\ldots,s]} := \left[\frac{A^{\mathcal{P}}}{A^{[i_s]}}\right] \in \mathsf{K}^{s\times m}.$$

We now need to determine if $v_s R^{[1,2,\ldots,s]} \in \mathsf{K}^{1\times m}$ is the zero vector, and find the index of the first nonzero element of $v_s R^{[1,2,\ldots,s]}$ otherwise: if we have precomputed a linear independence oracle T_s for $R^{[1,2,\ldots,s]}$ we can solve this problem in a Monte Carlo fashion in time $O(r \log m)$.

Since the situation in (2.1) is simply transposed v_s here, the computation in step 1 can be recast similarly by computing

$$\bar{v}_s := \left[\frac{1}{-(A^{\mathcal{P}}_{\mathcal{Q}})^{-1}\, b^{\mathcal{P}}}\right]^T \in \mathsf{K}^{1\times s}$$

and setting

$$\bar{R}^{[1,2,\ldots,s]} = \left[\ b\ \middle|\ A_{\mathcal{Q}}\ \right]^T \in \mathsf{K}^{s\times n}.$$

By Remark 4, $\bar{v}_s$ can be computed in $O(s)$ field operations.

This gives the following algorithm.

1. Choose $\alpha_1, \ldots, \alpha_{m-1}, \beta_1, \ldots, \beta_{n-1}$ uniformly and randomly from K.
2. Use algorithm `OracleSolve` described in Section 2, but instead of explicitly computing the vectors in (2.1) for $s = 1, 2, \ldots$, initialize T_0 to be an oracle tree for the $0 \times m$ vector, and use the method supporting Theorem 12 to construct from T_{s-1} and $R^{[s]}$ an oracle tree T_s for $R^{[1,2,\ldots,s]}$ based on $\alpha_1, \ldots, \alpha_{m-1}$. Similarly, instead of explicitly computing the vectors in (2.2), use a sequence of oracle trees based on $\beta_1, \ldots, \beta_{n-1}$.
3. Assay correctness of the output of the algorithm either by checking that $Ax = b$ or checking that $uA = 0$.

Corollary 15 bounds from below the probability that the linear independence oracles in step 2 will make exactly the same choices for $i_1, i_2, \ldots, i_s$ and $j_1, j_2, \ldots, j_s$ as the deterministic algorithm `OracleSolve` described in Section 2. We obtain the following result as a corollary of Lemma 10 (cost of using an oracle), Theorem 12 (constructing the sequences of oracles) and Corollary 15 (probability of success).

Theorem 16. *Let K be a finite field. There exists a randomized algorithm* `RandomOracleSolve`(A, b) *that takes as input $A \in \mathsf{K}^{n\times m}$ and $b \in \mathsf{K}^{n\times 1}$, and returns as output either*

- *"FAIL, x, $\mathcal{P}$, $\mathcal{Q}$" with $Ax \neq b$, or*
- *"FAIL, u, $\mathcal{P}$, $\mathcal{Q}$" with $uA \neq 0$ and $ub \neq 0$, or*
- *"consistent, x, $\mathcal{P}$, $\mathcal{Q}$" with $Ax = b$, or*
- *"inconsistent, u, $\mathcal{P}$, $\mathcal{Q}$" with $uA = 0$ and $ub \neq 0$.*

Here, $\mathcal{P} = [i_1, \ldots, i_s]$ and $\mathcal{Q} = [j_1, \ldots, j_s]$ are such that $A^{\mathcal{P}}$ and $A_{\mathcal{Q}}$ have full row and column rank s, respectively. The algorithm has the following properties:

1. *$n + m - 2$ random choices from K are required.*
2. *If r is the rank of A, then with probability at least*

$$\left(1 - \frac{r}{\#\mathsf{K}}\right)^{\lceil \log_2 n \rceil + \lceil \log_2 m \rceil}, \tag{5.1}$$

FAIL is not returned and the output is identical to that of algorithm `OracleSolver` *supporting Theorem 1.*

3. The running time is bounded by

$$2r^3 + O(r^2(\log n + \log m) + r(n+m))$$

and

$$2r^3 + O(r^2(\log n + \log m) + n + m + |A_{\mathcal{Q}}| \log n + |A^{\mathcal{P}}| \log m)$$

field operations in K.

If $\#\mathsf{K}$ is too small we can work over a small field extension of $\#\mathsf{K}$ to ensure positive probability of success. The following corollary is obtained from (5.1) by substituting $r \leq \min(n, m)$ and using the fact that for any $x > 0$ and $y \in \mathbb{Z}_{>0}$ we have $1 - xy \leq (1-x)^y$.

Corollary 17. *If*

$$\#\mathsf{K} \geq 2\min(n, m)(\lceil \log_2 n \rceil + \lceil \log_2 m \rceil)$$

then the probability that algorithm `RandomOracleSolve` *does not return FAIL is at least* $1/2$.

By Corollary 17, the degree of the field extension required to ensure a probability of success at least $1/2$ is bounded by $O(\log n + \log m)$ in the worst case. If a field extension is required, then the cost of constructing and using the sequence of oracles increase by a multiplicative factor that is softly linear in $\log n + \log m$. However, the inverse computations via rank one updates during each phase of the oracle solver are still performed over the ground field and have overall cost bounded by $2r^3 + O(r^2)$ field operations. This gives the following result, valid over any finite field.

Corollary 18. *There exists a Las Vegas algorithm for problem* LinSys *that has running time*

$$2r^3 + (r^2 + n + m + |A_{\mathcal{P}}| + |A_{\mathcal{Q}}|)^{1+o(1)},$$

where $\mathcal{P}$ *and* $\mathcal{Q}$ *are the indices of the subsets of at most* $r+1$ *rows and* r *columns of* A *that are examined by the algorithm.*

Chapter 6

Faster rank profiles

In the following theorem, let `ImprovedRankProfile` be identical to Algorithm 2 `RandomRankProfile` except with the call to the deterministic algorithm `OracleSolve` replaced with a call to the faster algorithm `RandomOracleSolve` supporting Theorem 16.

The following theorem follows as a corollary of Theorems 6 and 16.

Theorem 19. *Let* K *be a finite field. There exists a randomized algorithm* `ImprovedRankProfile`(A) *that takes as input* $A \in \mathsf{K}^{n\times m}$ *and returns as output* $s \in \mathbb{Z}_{\geq 0}$ *together with lists* $\mathcal{P} = [i_1, \ldots, i_s]$ *and* $\mathcal{C} = [j_1, \ldots, j_s]$ *such that* $A^{\mathcal{P}}$ *and* $A_{\mathcal{C}}$ *have full row and column rank* s*, respectively. The algorithm has the following properties:*

1. $n + 2m - 2$ *random choices from* K *are required.*
2. *If* r *is the rank of* A*, then with probability at least*

$$\left(1 - \frac{1}{\#\mathsf{K}}\right)^{r} \left(1 - \frac{r}{\#\mathsf{K}}\right)^{\lceil \log_2 n \rceil + \lceil \log_2 m \rceil},$$

   $\mathcal{P}$ *and* $\mathcal{C}$ *are the row and column rank profiles of* A*, respectively.*
3. *The running time is bounded by*

$$2r^3 + O(r^2(\log n + \log m) + nm)$$

   and

$$2r^3 + O(r^2(\log n + \log m) + n + m + |A| + |A_{\mathcal{C}}| \log n + |A^{\mathcal{P}}| \log m)$$

   field operations in K*.*

The following two corollaries are very similar to Corollaries 17 and 18.

Corollary 20. *If*

$$\#\mathsf{K} \geq 2\min(n,m)(1+\lceil\log_2 n\rceil+\lceil\log_2 m\rceil)$$

then the probability that `ImprovedRankProfile` *returns the correct result is at least* $1/2$.

Corollary 21. *There exists a Monte Carlo algorithm for problem* RankProfiles *that has running time bounded by* $2r^3+(r^2+n+m+|A|)^{1+o(1)}$.

Chapter 7

A Relaxed Algorithm for Online Matrix Inversion

Throughout this chapter let $A \in \mathsf{K}^{n \times n}$ and $b \in \mathsf{K}^{n \times 1}$ be given. Let A_s denote the leading $s \times s$ submatrix of A and $b_s \in \mathsf{K}^{s \times 1}$ be the vector containing the first s elements of b as shown in the following augmented system.

$$\left[\begin{array}{c|c} A & b \end{array}\right] = \left[\begin{array}{c|c} \boxed{A_s} & \boxed{b_s} \end{array}\right]$$

Assume A has generic rank profile, that is, A_s is nonsingular for $1 \leq s \leq n$. In this chapter we consider the following problem.

- OnlineSystem: Let $A \in \mathsf{K}^{n \times n}$ with generic rank profile and $b \in \mathsf{K}^{n \times 1}$ be given. Suppose the rows of the augmented system $\left[\begin{array}{c|c} A & b \end{array}\right]$ are given one at a time, from first to last. As soon as rows $1, 2, \ldots, s$ of $\left[\begin{array}{c|c} A & b \end{array}\right]$ are given, produce the subsystem solution $A_s^{-1} b_s$, for $s = 1, 2, \ldots, n$.

Because A is assumed to have generic rank profile, Gaussian elimination without pivoting produces a unique decomposition $A^{-1} = P_n \cdot P_{n-1} \cdots P_1$, where $P_s = R_s \cdot L_s$ is the product of a pair of structured matrices R_s and L_s, $1 \leq s \leq n$. The matrices R_s and L_s will be defined precisely in the next section. For now, consider the following example which shows

the structure of matrices in the decomposition for $n = 6$.

$$A^{-1} = \underbrace{\overset{R_6}{\begin{bmatrix} 1 & & & & & * \\ & 1 & & & & * \\ & & 1 & & & * \\ & & & 1 & & * \\ & & & & 1 & * \\ & & & & & * \end{bmatrix}} \cdot \overset{L_6}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ * & * & * & * & * & 1 \end{bmatrix}}}_{P_6} \cdot \underbrace{\overset{R_5}{\begin{bmatrix} 1 & & & & * & \\ & 1 & & & * & \\ & & 1 & & * & \\ & & & 1 & * & \\ & & & & * & \\ & & & & 1 & \end{bmatrix}} \cdot \overset{L_5}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ * & * & * & * & 1 & \\ & & & & & 1 \end{bmatrix}}}_{P_5} \cdots \underbrace{\overset{R_1}{\begin{bmatrix} * & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}} \cdot \overset{L_1}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}}}_{P_1} \tag{7.1}$$

Moreover, $P_s P_{s-1} \cdots P_1 = \mathrm{diag}(A_s^{-1}, I_{n-s})$ for $1 \leq s \leq n$. Thus, computing the sequence

$$A_1^{-1} b_1,\ A_2^{-1} b_2,\ A_3^{-1} b_3, \ldots, A_n^{-1} b_n \tag{7.2}$$

can be accomplished by computing the sequence

$$P_1 b,\ P_2 P_1 b,\ P_3 P_2 P_1 b, \ldots,\ P_n \cdots P_3 P_2 P_1 b. \tag{7.3}$$

This motivates the definition of the following problem.

- OnlineInverse: Suppose the rows of an $A \in \mathsf{K}^{n \times n}$ with generic rank profile are given one at a time, from first to last. As soon as rows $1, 2, \ldots, s$ of A are given, the pair of matrices $P_s = R_s \cdot L_s$ should be produced, for $s = 1, 2, \ldots, n$.

On the one hand, if the pairs of structured matrices $P_1, P_2, \ldots, P_n$ are known, the sequence shown in (7.3) can be computed in $O(n^2)$ field operations. On the other hand, a lower bound on the cost of solving problem OnlineInverse is $\Omega(n^2)$ since this a lower bound for the total size (number of field elements) to write down the sequence $P_1, P_2, \ldots, P_n$. It follows that any algorithm for problem OnlineInverse immediately gives an algorithm for problem OnlineSystem that supports the same running time bound. For the rest of this chapter, we consider algorithms to solve problem OnlineInverse.

7.1 Full inverse decomposition

For $1 \leq s \leq n$, let

$$A_s = \left[\begin{array}{c|c} A_{s-1} & u_s \\ \hline v_s & d_s \end{array}\right] \in \mathsf{K}^{s \times s},$$

where $u_s \in \mathsf{K}^{(s-1)\times 1}$, $v_s \in \mathsf{K}^{1\times(s-1)}$, and $d_s \in \mathsf{K}$. Suppose we have computed a decomposition of A_{s-1}^{-1} for some $s > 0$. Then Gaussian elimination produces a pair of $s \times s$ matrices $\bar{L}_s$ and $\bar{R}_s$ such that

$$\underbrace{\left[\begin{array}{c|c} I_{s-1} & \begin{matrix} \bar{R}_s \\ -A_{s-1}^{-1}u_s \end{matrix} \\ \hline & (d_s - v_s A_{s-1}^{-1} u_s)^{-1} \end{array}\right]}_{A_s^{-1}} \left[\begin{array}{c|c} I_{s-1} & \begin{matrix} \bar{L}_s \\ \end{matrix} \\ \hline -v_s & 1 \end{array}\right] \left[\begin{array}{c|c} A_{s-1}^{-1} & \\ \hline & 1 \end{array}\right] \left[\begin{array}{c|c} A_{s-1} & \begin{matrix} A_s \\ u_s \end{matrix} \\ \hline v_s & d_s \end{array}\right] = \left[\begin{array}{c|c} I_{s-1} & \begin{matrix} I_s \\ \end{matrix} \\ \hline & 1 \end{array}\right]. \quad (7.4)$$

The exact formula for A_s^{-1} is given by Lemma 2. However, the algorithms we describe in this chapter do not compute A_s^{-1} explicitly at each stage, but rather keep it as the product of pairs of structured matrices. For example, applying (7.4) for $s = 1, 2, \ldots, n$ gives the following full decomposition for the inverse of $A = A_n$.

$$A_n^{-1} = P_n \cdot \left[\begin{array}{c|c} A_{n-1}^{-1} & \\ \hline & 1 \end{array}\right] = P_n \cdot P_{n-1} \cdot \left[\begin{array}{c|c} A_{n-2}^{-1} & \\ \hline & I_2 \end{array}\right] = \cdots = P_n \cdot P_{n-1} \cdots P_1,$$

where $P_s \in \mathsf{K}^{n\times n}$, $1 \le s \le n$, is the product of two structured matrices:

$$P_s = R_s \cdot L_s = \left[\begin{array}{c|c} \bar{R}_s & \\ \hline & I_{n-s} \end{array}\right] \cdot \left[\begin{array}{c|c} \bar{L}_s & \\ \hline & I_{n-s} \end{array}\right] = \left[\begin{array}{ccc} I_{s-1} & \square & \\ & & I_{n-s} \end{array}\right] \cdot \left[\begin{array}{ccc} I_{s-1} & & \\ \square & 1 & \\ & & I_{n-s} \end{array}\right], \quad (7.5)$$

with $\bar{R}_s$ and $\bar{L}_s$ as in (7.4). Thus R_s and L_s are the identity matrices, except for possibly the column vector and row vector indicated by the rectangles in R_s and L_s, respectively.

Naturally, equation (7.4) gives an iterative approach to solve problem OnlineInverse. Once the matrix×vector product $A_{s-1}^{-1}u_s$ has been computed, the pair $P_s = R_s \cdot L_s$ can be computed in a further $O(s)$ field operations. Computing $A_{s-1}^{-1}u_s$ is equivalent to premultiplying the leading principal $(s-1) \times (s-1)$ submatrices of $P_1, P_2, \cdots, P_{s-1}$ by u_s sequentially at a cost of $4s^2 + O(s)$ field operations. Overall, all P_s, $1 \le s \le n$, can be computed sequentially in $2n^3 + O(n^2)$ field operations. In the next 2 sections we show how to incorporate matrix multiplication to solve problem OnlineInverse in overall time $O(n^w)$.

7.2 Relaxed inverse decomposition

To store each A_s^{-1} explicitly as a dense $s \times s$ matrix at each stage is too expensive, but the representation $\mathrm{diag}(A_s^{-1}, I_{n-s}) = P_s \cdot P_{s-1} \cdots P_1$ as the product of s pairs of structured

matrices is too lazy: both of these approaches lead to an algorithm with running time $\Omega(n^3)$. In this section, we present a relaxed inverse decomposition for A_s^{-1} such that, at stage s, after the first s rows of A are given, A_s^{-1} is represented as the product of $\mathrm{HammingWeight}(s) \leq \lceil \log s \rceil$ pairs of structured matrices, where $\mathrm{HammingWeight}(s)$ is the number of 1s in the binary representation of s.

Lemma 22. *$(R_j \cdot L_j) \cdot (R_{j-1} \cdot L_{j-1}) \cdots (R_i \cdot L_i)$ can be expressed as the product of two structured matrices:*

$$R_{j\sim i} \cdot L_{j\sim i} = \begin{bmatrix} I_{i-1} & & & \\ & \square & \cdots & \square \\ & & & I_{n-j} \end{bmatrix} \cdot \begin{bmatrix} I_{i-1} & & & \\ & \square & & 1 \\ & \vdots & \ddots & \\ & \square & & 1 \\ & & & I_{n-j} \end{bmatrix}, \tag{7.6}$$

where the column dimension of the submatrix indicated by the rectangle in $R_{j\sim i}$ and the row dimension of the submatrix indicated by the rectangle in $L_{j\sim i}$ are both $j-i+1$.

Proof. Consider the block case of (7.4):

$$\begin{bmatrix} I_{s-1} & -A_s^{-1}U \\ & (D - VA_s^{-1}U)^{-1} \end{bmatrix}_{\bar{R}_{j\sim i}} \begin{bmatrix} I_{s-1} & \\ -V & I_{j-i+1} \end{bmatrix}_{\bar{L}_{j\sim i}} \begin{bmatrix} A_{s-1}^{-1} & \\ & I_{j-i+1} \end{bmatrix} \begin{bmatrix} A_{s-1} & U \\ V & D \end{bmatrix}_{A_{s+j-i}} = \begin{bmatrix} I_{s-1} & \\ & I_{j-i+1} \end{bmatrix}_{I_{s+j-i}}$$

where $U \in \mathsf{K}^{(s-1)\times(j-i+1)}$, $V \in \mathsf{K}^{(j-i+1)\times(s-1)}$, and $D \in \mathsf{K}^{(j-i+1)\times(j-i+1)}$. By augmenting $\bar{R}_{j\sim i}$ as $\mathrm{diag}(\bar{R}_{j\sim i}, I_{n-s-j+i})$ and $\bar{L}_{j\sim i}$ as $\mathrm{diag}(\bar{L}_{j\sim i}, I_{n-s-j+i})$ we get the two structured matrices in (7.6). Thus we have

$$(R_{j\sim i}\, L_{j\sim i})\, \mathrm{diag}(A_{s-1}, I_{n-s+1})\, A = I_n$$

and

$$(R_j\, L_j R_{j-1}\, L_{j-1} \cdots R_i\, L_i)\, \mathrm{diag}(A_{s-1}, I_{n-s+1})\, A = I_n$$

The result follows by the uniqueness of the inverse. □

Note that Lemma 22 does not hold for general matrices $(R_* \cdot L_*)$ of the same shape in (7.5). But it does hold in the case where the pairs $(R_* \cdot L_*)$ are coming from Gaussian elimination directly.

Definition 23. *Let $P_{j\sim i}$ be the unevaluated product of the pair of matrices $R_{j\sim i} \cdot L_{j\sim i}$ with the shape as shown in (7.6). The* size *of $P_{j\sim i}$ is $j-i+1$, the number of pairs in the product.*

Note that when $i = 1$, $L_{j\sim i} = I_n$ and thus $P_{j\sim i} = R_{j\sim i}$ is explicitly computed. We do not give the exact formula for general $P_{j\sim i}$, but we will give the formula for a special case later. To have a basic idea we remark that the submatrix indicated by the rectangle in $L_{j\sim i}$ is equal to $-A^{[i,\ldots,j]}_{[1\ldots,i-1]}$. The following example gives the formula for a $P_{j\sim i}$ of size 2.

Example 24. *Let $\bar{P}_j$ and $\bar{P}_{j-1}$ be the leading $j \times j$ principal submatrix of P_j and P_{j-1}. Let*

$$\bar{P}_j = \begin{bmatrix} I_{j-2} & & \\ & 1 & \boxed{\begin{matrix} a \\ b \\ c \end{matrix}} \\ & & \end{bmatrix} \cdot \begin{bmatrix} I_{j-2} & & \\ & 1 & \\ & \boxed{\begin{matrix} d & e \end{matrix}} & 1 \end{bmatrix}, \bar{P}_{j-1} = \begin{bmatrix} I_{j-2} & \boxed{\begin{matrix} f \\ g \end{matrix}} & \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} I_{j-2} & & \\ \boxed{h} & 1 & \\ & & 1 \end{bmatrix}$$

Let $M = \begin{bmatrix} d & e \end{bmatrix} \begin{bmatrix} f & g \end{bmatrix}^T$. Then $\bar{P}_{j\sim j-1}$ can be expressed as the product of two structured matrices as follows:

$$\bar{P}_{j\sim j-1} = \bar{R}_{j\sim j-1} \cdot \bar{L}_{j\sim j-1} = \left[\begin{array}{cc|c} I_{j-2} & \begin{matrix} aM+f \\ bM+g \end{matrix} & \begin{matrix} a \\ b \end{matrix} \\ & \begin{matrix} cM \end{matrix} & \begin{matrix} c \end{matrix} \end{array}\right] \left[\begin{array}{ccc} I_{j-2} & & \\ \hline h & 1 & \\ \hline d & & 1 \end{array}\right]$$

From definition, $\mathrm{diag}(A_s^{-1}, I_{n-s}) = P_s \cdot P_{s-1} \cdots P_1$ is the product of s pairs of structured matrices of size 1. We now introduce a relaxed/lazy representation for $\mathrm{diag}(A_s^{-1}, I_{n-s})$, denoted by $(A_s)_L^{-1}$, that expresses A_s^{-1} as the product of at most $\log s$ pairs of structured matrices. We first give a few examples of the relaxed representation before giving a precise definition.

Example 25. *The relaxed representation of A_s^{-1} for $1 \le s \le 8$.*

s	$(A_s)_L^{-1}$
$1 = (1)_2$	$P_{1\sim 1}$
$2 = (10)_2$	$P_{2\sim 1}$
$3 = (11)_2$	$P_{3\sim 3} \cdot P_{2\sim 1}$
$4 = (100)_2$	$P_{4\sim 1}$
$5 = (101)_2$	$P_{5\sim 5} \cdot P_{4\sim 1}$
$6 = (110)_2$	$P_{6\sim 5} \cdot P_{4\sim 1}$
$7 = (111)_2$	$P_{7\sim 7} \cdot P_{6\sim 5} \cdot P_{4\sim 1}$
$8 = (1000)_2$	$P_{8\sim 1}$

The relaxed inverse decomposition for $s = 6, 7, 8$ are shown below. Note that for each of

these examples we assume $n = s$, to avoid having to augment with I_{n-s}.

$$(A_6)_L^{-1} = \underbrace{\overset{R_{6\sim5}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & * \\ & & 1 & & & * \\ & & & 1 & & * \\ & & & & & * \\ & & & & & * \end{bmatrix}} \cdot \overset{L_{6\sim5}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & * & * & 1 \\ & & & * & * & & 1 \end{bmatrix}}}_{P_{6\sim5}} \cdot \underbrace{\overset{R_{4\sim1}}{\begin{bmatrix} * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & 1 & \\ & & & & & 1 \end{bmatrix}} \cdot \overset{L_{4\sim1}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}}}_{P_{4\sim1}}$$

$$(A_7)_L^{-1} = \underbrace{\overset{R_7}{\begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & * \\ & & 1 & & & & * \\ & & & 1 & & & * \\ & & & & 1 & & * \\ & & & & & 1 & * \\ & & & & & & * \end{bmatrix}} \cdot \overset{L_7}{\begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ * & * & * & * & * & * & 1 \end{bmatrix}}}_{P_7} \cdot \underbrace{\overset{R_{6\sim5}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & * \\ & & 1 & & & * \\ & & & 1 & & * \\ & & & * & * & \\ & & & * & * & 1 \end{bmatrix}} \cdot \overset{L_{6\sim5}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & * & * & 1 \\ & & & * & * & & 1 \end{bmatrix}}}_{P_{6\sim5}} \cdot \underbrace{\overset{R_{4\sim1}}{\begin{bmatrix} * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & & \\ * & * & * & * & 1 & \\ & & & & & 1 \end{bmatrix}} \cdot \overset{L_{4\sim1}}{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}}}_{P_{4\sim1}}$$

$$(A_8)_L^{-1} = \underbrace{\overset{R_{8\sim1}}{\begin{bmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix}} \cdot \overset{L_{8\sim1}}{\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix}}}_{P_{8\sim1}}$$

To understand the following formal definition of the relaxed inverse, consider the case $s = 7$ and $\mathrm{diag}(A_7^{-1}, I_{n-7})$. The full inverse decomposition is

$$\mathrm{diag}(A_7^{-1}, I_{n-7}) = P_7 \cdot P_6 \cdot P_5 \cdot P_4 \cdot P_3 \cdot P_2 \cdot P_1$$

while the relaxed/lazy representation is

$$\mathrm{diag}(A_7^{-1}, I_{n-7}) = P_{7\sim7} \cdot P_{6\sim5} \cdot P_{4\sim1}.$$

The structure of the relaxed decomposition is determined as follows. The highest power of 2 that divides 7 is 4, and thus in the relaxed representation the rightmost four pairs $P_4 \cdot P_3 \cdot P_2 \cdot P_1$ of size 1 are compressed into the single pair $P_{4\sim1}$ of size 4. Continuing, the highest power of 2 that divides $7 - 4$ is 2, so $P_6 \cdot P_5$ is compressed to $P_{6\sim5}$. Finally, the highest power of 2 that divides $7 - 4 - 2$ is 1, so $P_7 = P_{7\sim7}$ is left alone.

Definition 26. *The relaxed/lazy representation of the inverse $(A_k)^{-1}$ is defined as*

$$(A_k)_L^{-1} = \begin{cases} (P_{k\sim2^m+1}) \cdots (P_{2^i+2^j\sim2^i+1}) \cdot (P_{2^i\sim1}), & \text{if } k \text{ is even;} \\ (P_k) \cdot (P_{k-1\sim2^m+1}) \cdots (P_{2^i+2^j\sim2^i+1}) \cdot (P_{2^i\sim1}), & \text{if } k \text{ is odd.} \end{cases}$$

where $i = \lfloor \log_2 k \rfloor$, $j = \lfloor \log_2(k - 2^i) \rfloor, \ldots$, *until the smallest possible number* $m = \lfloor \log_2(k - 2^i - 2^j - \cdots) \rfloor \geq 1$ *is reached. The first pair of* $(A_k)^{-1}$ *is* $P_{k \sim 2^m+1}$ *if* k *is even and* P_k *if* k *is odd.*

Note that when k is a power of two, we have $i = \log_2 k$ and

$$(A_k)_L^{-1} = P_{2^i \sim 1} = R_{2^i \sim 1} \cdot L_{2^i \sim 1} = R_{2^i \sim 1},$$

so A_k^{-1} is explicitly computed in this case. Otherwise, $(A_k)_L^{-1}$ is represented as the product of a sequence of pairs of structured matrices $P_{* \sim *}$. Observe that in Definition 26 we have $m < \cdots < j < i$. Thus the sizes of the product factors are strictly increasing from left to right.

In the next section we will give algorithms for the online computation of the sequence $(A_1)_L^{-1}, (A_2)_L^{-1}, (A_3)_L^{-1}, \ldots$. It turns out — and may already be clear from Example 25 — that the construction of $(A_s)_L^{-1}$ from $(A_{s-1})_L^{-1}$ and $P_{s \sim s}$ may require the combinations of two adjacent $(P_{* \sim *})$ of equal size. In particular, we will need to compute the compression

$$P_{j \sim j-2m+1} = P_{j \sim j-m+1} \cdot P_{j-m \sim j-2m+1}, \tag{7.7}$$

where both $P_{j \sim j-m+1}$ and $P_{j-m \sim j-2m+1}$ are of size m. Since this computation is important in deriving the cost of the relaxed approach, we give the formula and derive the cost for computing the left hand side of (7.7) given the two pairs on the right. For brevity, let $P_2 = R_2 \cdot L_2$, $P_1 = R_1 \cdot L_1$ and $P = R \cdot L$ denotes the leading $j \times j$ principal submatrix of $P_{j \sim j-m+1}$, $P_{j-m \sim j-2m+1}$ and $P_{j \sim j-2m+1}$ respectively. (Note that we are "overloading" the notation $P_2, R_2, L_2, \ldots$, but only temporarily in this example.) Then (7.7) is the block case of Example 24. We have

$$\underbrace{\overset{R_2}{\begin{bmatrix} I_{j-m} & \bar{R}_2 \end{bmatrix}} \cdot \overset{L_2}{\begin{bmatrix} I_{j-m} & & \\ & \bar{L}_2 & \\ & & 1 \\ & & & \ddots \\ & & & & 1 \end{bmatrix}}}_{P_2} \cdot \underbrace{\overset{R_1}{\begin{bmatrix} I_{j-2m} & \bar{R}_1 & \\ & & I_i \end{bmatrix}} \cdot \overset{L_1}{\begin{bmatrix} I_{j-2m} & & & \\ & \bar{L}_1 & & \ddots \\ & & & 1 \\ & & & & I_i \end{bmatrix}}}_{P_1}$$

$$= \underbrace{\overset{R}{\begin{bmatrix} I_{j-2m} & \bar{R} \end{bmatrix}} \cdot \overset{L}{\begin{bmatrix} I_{j-2m} & & \\ & \bar{L} & \\ & & 1 \\ & & & \ddots \\ & & & & 1 \end{bmatrix}}}_{P}$$

where

$$\bar{L} = \left[\ \bar{L}_1 \ \middle|\ (\bar{L}_2)_{[1,\ldots,j-2m]}\ \right]^T \in \mathsf{K}^{2m\times(j-2m)} \tag{7.8}$$

and $\bar{R} \in \mathsf{K}^{j\times 2m}$ can be computed by solving $R = (R_2L_2R_1L_1)L^{-1}$ to obtain

$$\bar{R} = \left[\ \bar{R}_2(\bar{L}_2\bar{R}_1) + \left[\ \bar{R}_1 \ \middle|\ 0\ \right]^T \ \middle|\ \bar{R}_2\ \right] \tag{7.9}$$

Theorem 27. *There exists a deterministic algorithm* `EqualSizeCompress` *that takes as input the two pairs* $P_{j\sim j-m+1}, P_{j-m\sim j-2m+1} \in \mathsf{K}^{n\times n}$*, both with size* m*, and returns as output the single pair* $P_{j\sim j-2m+1} \in \mathsf{K}^{n\times n}$ *of size* $2m$ *such that* $P_{j\sim j-2m+1} = P_{j\sim j-m+1}\cdot P_{j-m\sim j-2m+1}$*. The cost of the algorithm is* $O(nm^{\omega-1})$ *field operations in* K*.*

Proof. See (7.8) and (7.9) for the formula for computing the pair $P_{j\sim j-2m+1}$. Computing L is free since $\bar{L}$ can be read off from $\bar{L}_2$ and $\bar{L}_1$ directly. The dominating cost of computing R is to compute $\bar{R}_2(\bar{L}_2\bar{R}_1)$, where $\bar{R}_2 \in \mathsf{K}^{j\times m}$, $\bar{L}_2 \in \mathsf{K}^{m\times(j-m)}$ and $\bar{R}_1 \in \mathsf{K}^{(j-m)\times m}$. The product of two $m \times m$ matrix can be computed in cm^ω field operations, for some fixed constant c. Dividing $\bar{R}_2$, $\bar{L}_2$, and $\bar{R}_1$ into at most $\lceil j/m\rceil$ blocks of dimension bounded by $m \times m$, $\bar{L}_2\bar{R}_1$ can be computed in $\lceil j/m\rceil cm^\omega + O(mj)$ field operations and $\bar{R}_2(\bar{L}_2\bar{R}_1)$ takes another $\lceil j/m\rceil cm^\omega + O(mj)$ field operations. The result now follows by noting that $j \le n$. □

7.3 A relaxed algorithm for online matrix inversion

The iterative approach to solve problem OnlineInverse has overall cost $2n^3+O(n^2)$ field operations in K. We show in this section, how to incorporate matrix multiplication to solve the problem OnlineInverse in $O(n^w)$ field operations in K. We adopt two ideas used in relaxed [14] and online [12] algorithms.

The first idea is to *relax*, that is, use the relaxed representation $(A_k)_L^{-1}$ for A_k^{-1} as defined in Definition 26. The representation $(A_k)_L^{-1}$ for $k = 1, 2, \ldots, n$ is constructed in an incremental fashion. Let $k > 0$ and suppose $(A_{k-1})_L^{-1}$ and P_k are known. Since $(A_k)^{-1} = P_k(A_{k-1})_L^{-1}$, the relaxed representation $(A_k)_L^{-1}$ is computed by compressing the pair P_k (of size one) and the first pair of $(A_{k-1})_L^{-1}$ into a single pair if they have equal size, repeating if required. Algorithm 3 outlines the algorithm for computing $(A_k)_L^{-1}$ from $(A_{k-1})_L^{-1}$ and P_k.

Note that when k is odd, then by Definition 26 the first pair of $(A_{k-1})_L^{-1}$ has size greater than one, so the while loop is never executed. For even values of k the while loop is executed at least once, possibly as many as $\log k$ times.

Algorithm 3 `RelaxedRepresentation`$((A_{k-1})_L^{-1}, P_k)$

Require: $(A_{k-1})_L^{-1}, P_k \in \mathsf{K}^{n\times n}$
Ensure: $(A_k)_L^{-1}$
1: $P := P_k$;
2: $P_l :=$ first pair in $(A_{k-1})_L^{-1}$;
3: **while** (size of P = size of P_l) **do**
4: $\quad P :=$ `EqualSizeCompress`(P, P_l);
5: $\quad$ Remove P_l from $(A_{k-1})_L^{-1}$;
6: $\quad P_l :=$ first pair of $(A_{k-1})_L^{-1}$;
7: **end while**
8: **return** $P \cdot (A_{k-1})_L^{-1}$;

Example 28. *The steps of the call to* `RelaxedRepresentation`$((A_7)_L^{-1}, P_8)$ *to compute the relaxed representation* $(A_8)_L^{-1}$ *are as follows.*

$$\begin{aligned}(A_8)_L^{-1} &= P_8 \cdot (A_7)_L^{-1} \\ &= P_8 \cdot (P_7 \cdot P_{6\sim5} \cdot P_{4\sim1}) \\ &= P_{8\sim7} \cdot P_{6\sim5} \cdot P_{4\sim1} && (7.10)\\ &= P_{8\sim5} \cdot P_{4\sim1} && (7.11)\\ &= P_{8\sim1} && (7.12)\end{aligned}$$

Three compressions are required: the compression $P_{8\sim7} :=$ `EqualSizeCompress`$(P_{8\sim8}, P_{7\sim7})$ *of size 1 to 2 in (7.10); the compression* $P_{8\sim5} :=$ `EqualSizeCompress`$(P_{8\sim7}, P_{6\sim5})$ *of size 2 to 4 in (7.11); and the compression* $P_{8\sim1} :=$ `EqualSizeCompress`$(P_{8\sim5}, P_{4\sim1})$ *of size 4 to 8 in (7.12).*

The second idea is to *anticipate* computations. To make it clear that rows of A are given one at a time, we use a work matrix B for the anticipated computations. B is initialized to be the $n \times n$ zero matrix. At stage $s > 0$, row s of A is copied to row s of B. At stage s the first s rows of the input matrix A are defined, and the (untransformed) input matrix

can be decomposed as follows, where u_s is the first column of U.

$$A = \left[\begin{array}{c|c:c} A_{s-1} & u_s & U \\ \hline v_s & d_s & \\ \hline \multicolumn{3}{c}{0} \end{array}\right] \in \mathsf{K}^{n\times n} \tag{7.13}$$

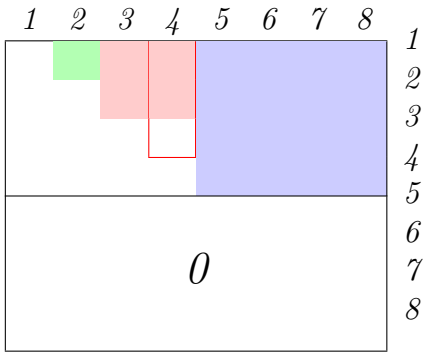
Recall that the dominant cost of computing the pair $P_s = R_s \cdot L_s$ arises from computing the matrix×vector product $A_{s-1}^{-1}u_s$ as shown in (7.5). At stage $s-1$, when $(A_{s-1})_L^{-1}$ has been computed, we don't apply A_{s-1}^{-1} to the single column u_s of A, which would be sufficient to compute the pair P_s at the next stage. Rather, we anticipate computations by applying the first pair P of the lazy representation $(A_{s-1})_L^{-1}$ to m columns of the work matrix B, where m is the size of P. This effectively incorporates matrix multiplication, and has the effect that at the beginning of stage s we have $B_{[s]}^{[1,\ldots,s-1]} = (A_{s-1})^{-1}u_s$. Algorithm 4 outlines the algorithm to compute P_k for a given k along with the anticipated computations. It first checks whether P_k exists, and reports "FAIL" if not. Detection is simple: if $d_k - v_k(A_{k-1})^{-1}u_k = 0$ then P_k does not exist.

Algorithm 4 `ComputeP`$(A, (A_{k-1})_L^{-1}, k)$

Require: $A \in \mathsf{K}^{n\times n}$ full rank, $(A_{k-1})_L^{-1}, k \in \mathbb{Z}_{\geq 1}$
Ensure: $P_k \in \mathsf{K}^{n\times n}$

```
Copy row k of A to row k of work matrix B;
if (d_k − v_k B_[k]^[1,...,k−1] = 0) then
    return "FAIL";
end if
Compute P_k := R_k · L_k using Equation (7.5);
if (k < n) then
    (A_k)_L^{-1} := RelaxedRepresentation((A_{k−1})_L^{-1}, P_k);
    Let C_k := P_{k∼*} be the first element of (A_k)_L^{-1} and s be the size of C_k;
    t := min(k + s, n);
    B_[k+1,...,t] := C_k B_[k+1,...,t];
end if;
return P_k;
```

Example 29. *We consider the computations of the first four stages of the relaxed online inverse update for* $A \in \mathsf{K}^{8\times 8}$. *The following shows the work matrix* B *with certain submatrices highlighted.*



At stage $s = 0$, B *is initialized to be the* 8×8 *zero matrix. For* $1 \le s \le 4$, *the computations done at each stage are summarized below.*

1. *Copy row* 1 *of* A *to row* 1 *of* B *and compute* $P_1 = R_1 \cdot L_1$.
   Apply P_1 *to column 2 of* B *(■ area).*
2. *Copy row* 2 *of* A *to row* 2 *of* B *and compute* $P_2 = R_2 \cdot L_2$.
   Compress $P_2 \cdot P_1 = P_{2\sim 1}$.
   Apply $P_{2\sim 1}$ *to columns 3, 4 of* B *(■ area).*
3. *Copy row* 3 *of* A *to row* 3 *of* B *and compute* $P_3 = R_3 \cdot L_3$.
   Apply P_3 *to column 4 of* B *(□ area).*
4. *Copy row* 4 *of* A *to row* 4 *of* B *and compute* $P_4 = R_4 \cdot L_4$.
   Compress $P_4 \cdot P_3 = P_{4\sim 3}$.
   Compress $P_{4\sim 3} \cdot P_{2\sim 1} = P_{4\sim 1}$.
   Apply $P_{4\sim 1}$ *to columns 5,6,7,8 of* B *(■ area).*

At stages $2^k = 1, 2, 4, \ldots$ *the explicit inverse has been computed since* $L_{2^k\sim 1} = I_n$ *and* $R_{2^k\sim 1} = \operatorname{diag}(A_{2^k}^{-1}, n - 2^k)$. *At the end of stage 3, though, we only have the relaxed representation* $\operatorname{diag}(A_3^{-1}, n-3) = P_3 \cdot P_{2\sim 1}$.

The relaxed algorithm to solve problem OnlineInverse is thus to call Algorithm 4 `ComputeP` n times with input $k = 1, 2, \ldots, n$, and the initial input $(A_0)_L^{-1} = I_n$.

Theorem 30. *There exists a relaxed algorithm* `RelaxedInverse` *to solve problem* OnlineInverse *in* $O(n^\omega)$ *field operations from* K.

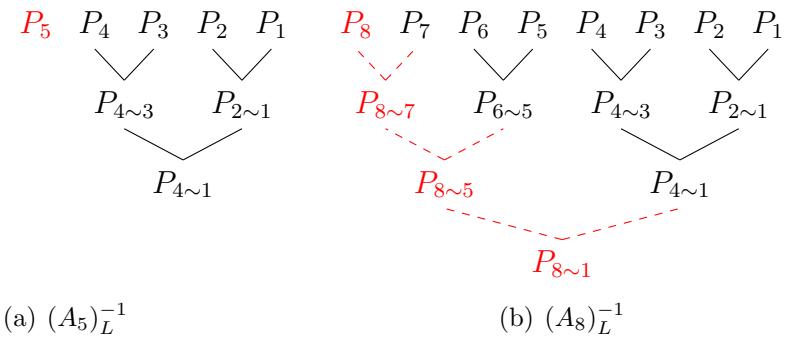
Proof. For simplicity, assume n is a power of two; otherwise, we can augment A as $\mathrm{diag}(A, I_*)$. There are two parts of the cost: (1) computing P_k and anticipation; (2) updating $(A_k)_L^{-1}$ from $(A_{k-1})_L^{-1}$ and P_k. For cost (1), the dominate cost is to anticipate computations (step 8 of Algorithm 4). At each stage k, step 8 costs $O(k \cdot s^{w-1})$ field operations from K, where s is the size of the first element of $(A_k)_L^{-1}$. From Definition 26, s is the highest power of 2 dividing k. Note for $k = n$, no anticipation is needed. The sequence of s for the first $n-1$ stages are listed below.

$$(2^{\nu_2(i)})_{1\le i\le n-1} = 1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1, 16, \cdots, 1, 2, 1$$

where $\nu_2(i)$ is the exponent of the largest power of 2 dividing i [21]. Note the number 2^i in the sequence appears $n/2^{i+1}$ times. For instance, 1 appears every other number, 2 appears every four numbers,$\cdots$. For $k \le n$, we obtain the following.

$$\begin{aligned}
\sum_{i=0}^{(\log n)-2} \frac{n}{2^{i+1}} \left(cn(2^i)^{\omega-1}\right) &= \frac{cn^2}{2} \sum_{i=0}^{(\log n)-2} 2^{i(\omega-2)} \\
&= \frac{cn^2}{2} \left(1 + 2^{\omega-2} + 4^{(\omega-2)} + 8^{(\omega-2)} + \cdots + (\frac{n}{4})^{\omega-2}\right) \\
&= \frac{cn^2}{2} \left(\frac{1 - 2^{(\omega-2)(\log n-1)}}{1 - 2^{\omega-2}}\right) \\
&= \frac{cn^2}{2} \left(\frac{1 - (n/2)^{\omega-2}}{1 - 2^{\omega-2}}\right) \\
&= \frac{cn^2}{2} O(n^{\omega-2}) \\
&= O(n^\omega)
\end{aligned}$$

For cost (2), the number of compressions done at stage k is equal to the maximal $c \in \mathbb{Z}$ such that $2^c \mid k$. Thus some stages are more costly than others. For example, when k is odd, $c = 0$, and thus no compression is performed. When k is a power of 2, then there are $\log k$ compressions required (see Figure 7.1, where compressions performed are highlighted in dashed lines). We consider the overall cost for $1 \le k \le n$, as shown in Figure 7.2. Nodes in the inverse tree are computed in a bottom-up fashion. We label level $l = 0, 1, \ldots, \log n$ from leaf up to root level. For $l > 0$, nodes at level l can be computed using Theorem 27



(a) $(A_5)_L^{-1}$ (b) $(A_8)_L^{-1}$

Figure 7.1: Examples of relaxed representation update

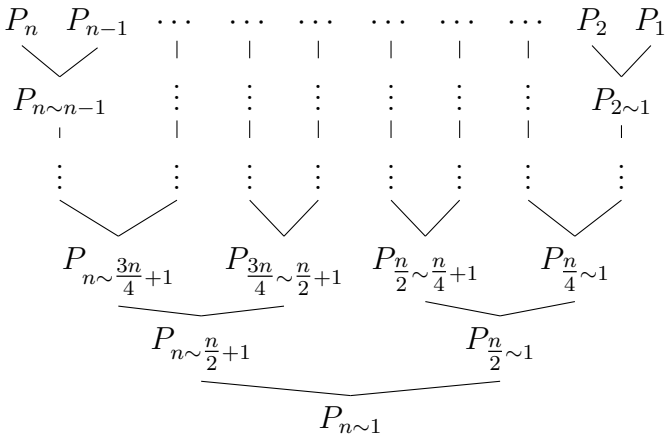


Figure 7.2: Online inverse tree

with $m = 2^{l-1}$. The number of nodes at level l is $n/2^l$. Overall, the cost of constructing the whole inverse tree is

$$\sum_{l=1}^{\log n} \left(cn(2^{l-1})^{w-1}\right) \frac{n}{2^l} = O(n^\omega)$$

The steps for achieving the result for this summation are skipped since it is similar to the previous summation. The result follows by adding the cost of both parts. □

Corollary 31. *There exists a deterministic algorithm to solve problem* OnlineSystem *in $O(n^\omega)$ field operations in* K.

Chapter 8

Application for RankProfiles

In this chapter we show how to use the algorithm `RelaxedInverse` supporting Theorem 30 to obtain a Monte Carlo algorithm for computing the row rank profile of an arbitrary $A \in \mathsf{K}^{n \times m}$ in time $(r^\omega + n + m + |A|)^{1+o(1)}$, where r is the rank of A.

The leading term $2r^3$ in the cost estimate of algorithm `ImprovedRankProfile` described in Chapter 6 arises from the iterative nature of computing, for $s = 1, 2, 3, \ldots$, the inverse $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} \in \mathsf{K}^{s \times s}$. However, considering the two steps of the oracle solver presented in Chapter 2, we can see that we do not need to compute $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1}$ explicitly at each stage. In particular, recall that the purpose of step 1 is to update $\mathcal{P}$ by setting i_s to be the first nonzero entry of $b - A_{\mathcal{Q}}(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} b^{\mathcal{P}}$. The main computation in step 1 (to allow use of a linear independence oracle) is thus to compute the linear system solution $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} b^{\mathcal{P}}$. Our approach will be to use algorithm `RelaxedInverse` to compute the vectors $(A_{\mathcal{Q}}^{\mathcal{P}})^{-1} b^{\mathcal{P}}$ at each stage via matrix multiplication. Now consider step 2. Recall that the purpose of step 2 is to update $\mathcal{Q}$ by setting j_s to be the first nonzero entry of $A^{[i_s]} - A_{\mathcal{Q}}^{[i_s]} (A_{\mathcal{Q}}^{\mathcal{P}})^{-1} A^{\mathcal{P}}$. Our approach will be to avoid the need to find j_s in step 2 by utilizing a Toeplitz preconditioner to ensure with high probability that j_s can simply be set to s.

Let $\mathcal{R}$ denotes the row rank profile of a matrix $A \in \mathsf{K}^{n \times m}$. In Section 8.1 we first consider the special case of an input matrix with full column rank and such that $A^{\mathcal{R}}$ has generic rank profile. In Section 8.2 we generalize to the case of an input matrix with full column rank but for which $A^{\mathcal{R}}$ does not necessarily have generic rank profile. Finally, in Section 8.3 we give an algorithm for the general case of an input matrix with unknown rank.

8.1 A has full column rank and $A^{\mathcal{R}}$ has generic rank profile

We first consider the special case when $A \in \mathsf{K}^{n\times m}$ has full column rank m and $A^{\mathcal{R}}$ has generic rank profile, that is, all the leading principal submatrices of $A^{\mathcal{R}}$ are nonsingular. Consider stage s of algorithm `ImprovedRankProfile` supporting Theorem 19. If, after the first step, $\mathcal{P} := [i_1, i_2, \ldots, i_s]$ has been correctly determined to be the first s entries of $\mathcal{R}$, then we can avoid any computation in step 2 by simply setting $j_s = s$. By Corollary 31, the vectors $(A^{\mathcal{P}}_{\mathcal{Q}})^{-1}b^{\mathcal{P}}$ in step 1 for stages $s = 1, 2, \ldots, m$ can be computed in overall time $O(m^\omega)$. In the following theorem, let `ModifiedRankProfile` be identical to `ImprovedRankProfile` except with the modifications just described.

Theorem 32. *Let* K *be a finite field. There exists a randomized algorithm* `ModifiedRankProfile` *that takes as input* $A \in \mathsf{K}^{n\times m}$ *of rank* m *and* $A^{\mathcal{R}}$ *has generic rank profile, where* $\mathcal{R}$ *is the row rank profile of* A*, and returns as output* $s \in \mathbb{Z}_{\geq 0}$ *together with list* $\mathcal{P} = [i_1, \ldots, i_s]$ *such that* $A^{\mathcal{P}}$ *has full row rank* s*. The algorithm has the following properties.*

1. $n + m - 1$ *random choices from* K *are required.*
2. *With probability at least*

$$\left(1 - \frac{1}{\#\mathsf{K}}\right)^m \left(1 - \frac{m}{\#\mathsf{K}}\right)^{\lceil \log_2 n \rceil},$$

   $\mathcal{P}$ *is equal to the row rank profile* $\mathcal{R}$ *of* A*.*
3. *The running time is bounded by* $(m^\omega + |A|)^{1+o(1)}$ *field operations in* K*.*

Proof. Compared to Theorem 19, there are fewer random choices because we only use the linear independence oracle on rows of A. The probability of success is slightly better accordingly. The cost estimate follows as a corollary of Theorem 19 and Corollary 31. □

8.2 A has full column rank

We now consider the case when $A \in \mathsf{K}^{n\times m}$ has full column rank m, but $A^{\mathcal{R}}$ does not necessarily have generic rank profile.

Lemma 33. *[16, Theorem 2]. Let* K *be a finite field. If* $v_2, v_3, \ldots, v_m \in \mathsf{K}$ *are chosen uniformly and randomly, then the lower triangular Toeplitz matrix*

$$L = \begin{bmatrix} 1 & & & & \\ v_2 & 1 & & & \\ v_3 & v_2 & 1 & & \\ \vdots & & \ddots & \ddots & \\ v_m & v_{m-1} & \cdots & v_2 & 1 \end{bmatrix} \in \mathsf{K}^{m \times m}$$

has the property that $A^{\mathcal{R}}L$ *has generic rank profile with probability at least* $1 - m(m+1)/(2\#\mathsf{K})$.

We remark that compared to [16, Theorem 2], the above lemma only uses one Toeplitz preconditioner L because the input matrix A has full column rank; this accounts for the extra factor 2 in the denominator of the probability estimate compared to [16, Theorem 2].

Lemma 34. *AL has the same row rank profile as the original matrix A.*

Proof. Suppose $U \in \mathsf{K}^{m \times m}$ is a nonsingular matrix that transforms A into reduced column echelon form C: $AU = C$. Then $(AL)(L^{-1}U) = C$ is also in reduced column echelon form, where $L^{-1}U$ is a nonsingular transformation. The result follows by the uniqueness of the reduced column echelon form. □

From Lemma 34, the list $\mathcal{P}$ returned from `ModifiedRankProfile`(AL) will be the row rank profile of the original matrix A with high probability. However, computing the product AL directly is expensive, and exceeds the running time bound of Theorem 32. Alternatively, we now describe a modification of algorithm `ModfiedRankProfile`, called `AdvancedRankProfile`, that takes as input $A \cdot L$ unevaluated, as a product of two matrices, and only evaluate parts of the product when necessary. We first observe that

$$(AL)^{\mathcal{P}} = A^{\mathcal{P}}L$$

and

$$(AL)_{\mathcal{Q}} = AL_{\mathcal{Q}}.$$

Thus the computation of (2.1) for finding the first nonzero entry of the column vector $b - A_{\mathcal{Q}} \cdot (A^{\mathcal{P}}_{\mathcal{Q}})^{-1} b^{\mathcal{P}}$ can be recast as follows.

$$b - (AL)_{\mathcal{Q}} \cdot ((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1} b^{\mathcal{P}} = b - A \cdot L_{\mathcal{Q}} \left(((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1} b^{\mathcal{P}} \right) \tag{8.1}$$

where $b = A(Lw)$ and $w \in \mathsf{K}^{m\times 1}$ is chosen uniformly and randomly. Using the expression on the right hand side of (8.1) allows us to build a linear independence oracle on the rows of A instead of $(AL)_{\mathcal{Q}}$. Moreover, the linear independence oracle is built at once at the beginning instead of incrementally, which will not affect the overall cost of constructing the oracle since A has full column rank. The remaining computation to consider in (8.1) is to compute $L_{\mathcal{Q}}\left(((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1}b^{\mathcal{P}}\right)$. Once the vector $((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1}b^{\mathcal{P}}$ has been computed, computing the product $L_{\mathcal{Q}}\left(((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1}b^{\mathcal{P}}\right)$ costs $\mathsf{M}(m)$ field operations, where $\mathsf{M}(m) = O(m\log m\log\log m)$ [1] is the cost of multiplication of polynomials with degree m. There are m stages in total. Overall, having the input $A\cdot L$ unevaluated yields an additional cost $m\,\mathsf{M}(m) = (m^2)^{1+o(1)}$ field operations to find the first nonzero entry in (8.1) compared to Theorem 32, which is within the cost bound of Theorem 32.

The vector $((AL)^{\mathcal{P}}_{\mathcal{Q}})^{-1}b^{\mathcal{P}}$ is updated using the relaxed approach as described in the previous chapter. At stage s, once the next nonzero row index i_s of (8.1) has been chosen, the corresponding row $(AL)^{[i_s]}$ is computed and copied to the work matrix B. Computing $(AL)^{[i_s]} = A^{[i_s]}L$ for $1 \le s \le m$ also has overall cost $m\,\mathsf{M}(m)$, and thus is within the cost bound of Theorem 32. We obtain the following result.

Theorem 35. *Let K be a finite field. There exists a randomized algorithm* `AdvancedRankProfile` *that takes as input $A \in \mathsf{K}^{n\times m}$ of full column rank m and returns as output $s \in \mathbb{Z}_{\ge 0}$ together with list $\mathcal{P} = [i_1,\ldots,i_s]$ such that $A^{\mathcal{P}}$ has full row rank s. The algorithm has the following properties.*

1. *$n + 2m - 2$ random choices from K are required.*
2. *With probability at least*

$$\left(1-\frac{1}{\#\mathsf{K}}\right)^{m}\left(1-\frac{m}{\#\mathsf{K}}\right)^{\lceil\log_2 n\rceil}\left(1-\frac{m(m+1)}{2\#\mathsf{K}}\right), \tag{8.2}$$

   $\mathcal{P}$ is the row rank profile of A.

3. *The running time is bounded by $(m^\omega + |A|)^{1+o(1)}$ field operations in K.*

Proof. Compared to Theorem 32, there are $m-1$ more random choices coming from the Toeplitz preconditioner (Lemma 33). The probability of success is slightly worse accordingly. The running time has been addressed as discussed above. □

If $\#\mathsf{K}$ is too small we can work over a small field extension of $\#\mathsf{K}$ to ensure positive probability of success.

Corollary 36. *If*

$$\#\mathsf{K} \geq 10m \max\left(\frac{1}{2}(m+1), \lceil \log_2 n \rceil + 1\right)$$

then the probability that algorithm `AdvancedRankProfile` *does not return FAIL is at least* $3/4$.

Proof. Using the fact that for any $x > 0$ and $y \in \mathbb{Z}_{>0}$ we have $1 - xy \leq (1-x)^y$, (8.2) $\geq 3/4$ can be simplified as

$$\underbrace{\left(1 - \frac{m(\lceil \log_2 n \rceil + 1)}{\#\mathsf{K}}\right)}_{(1)} \underbrace{\left(1 - \frac{m(m+1)}{2\#\mathsf{K}}\right)}_{(2)} \geq \frac{3}{4}.$$

The result follows by taking $(1) \geq \sqrt{3}/2$, $(2) \geq \sqrt{3}/2$, and $(1 - \sqrt{3}/2) > 0.1$. □

If a field extension is required, then the degree of the field extension is the minimal d such that

$$\left(1 - \frac{m(\lceil \log_2 n \rceil + 1)}{(\#\mathsf{K})^d}\right)\left(1 - \frac{m(m+1)}{2(\#\mathsf{K})^d}\right) \geq \frac{3}{4}.$$

By Corollary 36, d is bounded by $O(\log\log n + \log m)$ in the worst case to ensure a probability of success at least $3/4$. The additional cost of working over an extension field increase by a multiplicative factor that is softly linear in $\log\log n + \log m$. This gives the following result, valid over any finite field.

Corollary 37. *There exists a Monte Carlo algorithm that computes the row rank profile of a full column rank matrix in time* $(m^\omega + |A|)^{1+o(1)}$.

8.3 Arbitrary A with unknown rank

Now consider the general case for the input matrix $A \in \mathsf{K}^{n\times m}$ of (unknown) rank r. To compute the row rank profile of A, we use the following two randomized algorithms.

- The Monte Carlo rank algorithm in [4, Theorem 2.11] to find a subset of $s \leq r$ linearly independent columns of A in time $(r^\omega + |A|)^{1+o(1)}$. The algorithm should return $s = r$ with probability at least $3/4$. Provided $r = s$, the column space of the resulting matrix is equal to that of A so they must have the same row rank profile.

- The Monte Carlo algorithm supporting Corollary 36 and 37 with probability of correctness at least 3/4.

Since $1/4 + 1/4 = 1/2$, the probability of failure for using both algorithms is at most $1/2$. For the column rank profile, we can simply apply the same algorithm to the transpose of A. This gives the following result.

Corollary 38. *There exists a Monte Carlo algorithm for problem* RankProfiles *that has running time bounded by* $(r^\omega + |A|)^{1+o(1)}$.

Chapter 9

Conclusions and future work

We have given a Las Vegas algorithm for LinSys that has running time

$$2r^3 + (r^2 + n + m + |R| + |C|)^{1+o(1)},$$

where $|R|$ and $|C|$ are the number of nonzero entries in the rows and columns, respectively, that are examined. An open problem is to reduce the $2r^3$ term in this cost estimate by incorporating fast matrix multiplication.

We also give Monte Carlo algorithms for RankProfiles that have running times $2r^3 + (r^2 + n + m + |A|)^{1+o(1)}$ and $(r^\omega + n + m + |A|)^{1+o(1)}$, where $|A|$ is an upper bound on the number of nonzero entries in A. The latter cost estimate is achieved by incorporating a relaxed approach for online inverse update. It would be interesting to find other applications of our algorithm `RelaxedInverse`.

For convenience, we have assumed that K is a finite field, which allowed us to choose elements uniformly and randomly from K, but it should not be difficult to extend algorithms `RandomOracleSolve` and `ImprovedLinSys` to work over any field, for example by choosing random elements from a sufficiently large subset of K.

References

[1] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[2] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications*, 343–344:119–146, 2002. Special issue on *Infinite Systems of Linear Equations Finitely Specified*.

[3] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'05*, pages 92–99. ACM Press, New York, 2005.

[4] H. Y. Cheung, T. C. Kwok, and L. C. Lau. Fast matrix rank algorithms and applications. *Journal of the ACM*, 60(5):733–751, 2013. Article No. 31.

[5] J.-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In T. Mora, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'02*, pages 63–74. ACM Press, New York, 2002.

[6] J.-G. Dumas, C. Pernet, and Z. Sultan. Simultaneous computation of the row and column rank profiles. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'13*, pages 181–188. ACM Press, New York, 2013.

[7] J.-G. Dumas and G. Villard. Computing the rank of large sparse matrices over finite fields. *CASC'2002*, pages 47–62, 2002.

[8] W. Eberly. Early termination over small fields. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'03*, pages 80–87. ACM Press, New York, 2003.

[9] J.-C. Faugère. Parallelization of Grobner basis. In H. Hong, editor. *First International Symposium on Parallel Symbolic Computation, PASCO '94*, pages 124–132, 1994.

[10] J.-C. Faugère. A new efficient algorithm for computing Grobner basis (F4). *Journal of Pure and Applied Algebra*, 139:61–88, June 1999.

[11] M. Giesbrecht, A. Lobo, and B. D. Saunders. Certifying inconsistency of sparse linear systems. In O. Gloor, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'98*, pages 113—119. ACM Press, New York, 1998.

[12] P. Giorgi and R. Lebreton. Online order basis algorithm and its impact on block Wiedemann algorithm. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'14*. ACM Press, New York, 2014.

[13] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):12:1–12:25, May 2008.

[14] J. van der Hoeven. Relax, but don't be too lazy. *Journal of Symbolic Computation*, 36(6):479–542, 2002.

[15] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank-profile revealing Gaussian elimination and the CUP matrix decomposition. *Journal of Symbolic Computation*, 56:56–58, 2013.

[16] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proc. AAECC-9, Lecture Notes in Comput. Sci., vol. 539*, pages 29–38, 1991.

[17] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. *Advances in Cryptology-CRYPT0'90, Lecture Notes in Computer Science*, 537:109–133, 1991. http://www.research.att.com/~amo/doc/arch/sparse.linear.eqs.ps.

[18] J. P. May, B. D.Saunders, and Z. Wan. Efficient matrix rank computation with application to the study of strongly regular graphs. In J. P. May, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'07*, pages 277–284. ACM Press, New York, 2007.

[19] T. Mulders and A. Storjohann. Rational solutions of singular linear systems. In C. Traverso, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'00*, pages 242–249. ACM Press, New York, 2000.

[20] B.D. Saunders and B.S. Youse. Large matrix, small rank. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'09*, pages 317–324. ACM Press, New York, 2009.

[21] N. J. A. Sloane. The on-line encyclopedia of integer sequences. *Notices of the American Mathematical Society*, 50(8):912–915, 2003.

[22] W. Stein. *Modular Forms, a Computational Approach. Graduate Studies in Mathematics.* American Mathematical Society, 2007.

[23] A. Storjohan and S. Yang. Linear independence oracles and applications to rectangular and low rank linear systems. In *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC'14*. ACM Press, New York, 2014. To appear.

[24] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, Swiss Federal Institute of Technology, ETH–Zurich, 2000.

[25] A. Storjohann and T. Mulders. Fast algorithms for linear algebra modulo N. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms — ESA '98*, LNCS 1461, pages 139–150. Springer Verlag, 1998.

[26] A. Storjohann and S. Yang. A relaxed algorithm for online matrix inversion, 2014. Poster available at https://cs.uwaterloo.ca/~astorjoh/online.pdf.

[27] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1):3–35, 2001.

[28] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.

[29] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC'12*, pages 887–898. ACM, New York, 2012.

[30] R. Yuster. Generating a d-dimensional linear subspace efficiently. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 467–470. Society for Industrial and Applied Mathematics, 2010.