DEFINE ISSUE using \ issue

Exact Solutions to Linear Systems of Equations using Output Sensitive Lifting

Daniel Steffy[*]
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA, 30332-0205, USA
desteffy@gatech.edu

Abstract

Many methods have been developed to symbolically solve systems of linear equations over the rational numbers. A common approach is to use p-adic lifting or iterative refinement to build a modular or approximate solution, then apply rational reconstruction. An upper bound can be computed on the number of iterations these algorithms must perform before applying rational reconstruction. In practice such bounds can be conservative. Output sensitive lifting is the technique of performing rational reconstruction at intermediate steps of the algorithm, and verifying correctness. In this paper we show how using an appropriate output sensitive lifting strategy can improve several algorithms. We show this procedure to be computationally effective and introduce a new algorithm that directly combines rational reconstruction and iterative refinement by warm starting the Extended Euclidean Algorithm.

1 Introduction

Solving rational or integer linear systems of equations is a well developed area of symbolic computation. Dixon [8] gave an effective procedure for solving systems exactly by computing $\hat{x} = A^{-1}b \mod p^k$ through p-adic lifting and applying rational reconstruction to recover the exact rational solution. Wiedemann's black-box method for solving systems of equations over a finite field can also be used along with p-adic lifting or the Chinese Remainder Algorithm to solve systems in the sparse setting [15, 29]. Alternate techniques include calling a fixed precision numerical solver within an iterative refinement routine to find an extended precision solution $\hat{x} \approx A^{-1}b$, sufficient for rational reconstruction to be applied [25, 27]. Others have further developed these methods [5, 6, 10, 12, 19].

A core component of these techniques is rational number reconstruction, which allows an exact rational solution to be recovered from either a modular or approximate solution. We define the bitsize of a nonzero rational number p/q to be $\mathit{bitsize}(p/q) = \lceil \log(|pq|) \rceil$ and the bitsize of a rational vector to be the maximum bitsize over its components. In order to reconstruct a rational solution x from a modular solution, the system of equations must be solved modulo a number M, the size of which depends on $\mathit{bitsize}(A^{-1}b)$. Similarly, if a rational solution is to be reconstructed from an approximate solution, the level of approximation depends on $\mathit{bitsize}(A^{-1}b)$. The solution vector is unknown before solving the system so an upper bound on its size is computed to guide the rational reconstruction procedure. For an integer system of equations $Ax = b$, Cramer's rule and the Hadamard determinant bound imply that a solution has bitsize at most $\log(\|A\|_2^{2n-1}\|b\|_2)$. This bound can often be excessive, leading to unnecessary computation, both in the number of lifting loops that must be performed, and in the cost of performing rational reconstruction on large integers. *Output sensitive lifting* is the technique of attempting rational reconstruction at intermediate steps of the algorithm. The term output sensitive lifting is used in [5, 6] and is incorporated into their algorithms. The idea of output sensitive lifting has also been used in several other settings, such as the computation of determinants by Kaltofen [14] where it is referred to as early termination. Output sensitive lifting was also studied in [4] for solving systems of equations over cyclotomic fields. Use of output sensitive lifting can provide both theoretical and practical improvements when solving systems of equations exactly. It is applicable in both the dense and sparse settings.

The commonly used bounds can be weak for several reasons. Cramer's rule tells us that the denominator of a solution to an integer system $Ax = b$ will divide $det(A)$ and the Hadamard bound gives $det(A) \leq \|A\|_2^n$. While tight in some cases, the Hadamard bound is often weak, this is experimentally and probabilistically studied in [1]. Even if the determinant is well approximated by the Hadamard bound, or calculated exactly, it only provides an upper bound on the solution denominator size and there are many situations in which solutions will not meet this bound. Systems of equations may have special structure leading to small solution size, or integral solutions. In [7] it was found that in systems of equations arising from linear programming applications, the solution bitsize was often much lower than this bound. In such cases, application of output sensitive lifting has a huge impact on solution times.

The bitsize of the solution to a system of equations also depends on the right hand side. A matrix which has very complex solutions for particular right hand side vectors will have trivial or uncomplicated solutions for others. This is one way in which exact precision linear algebra differs significantly from numerical linear algebra. If a matrix can be successfully factored or inverted numerically, then solving the system for different right hand sides, represented in machine precision, will require almost identical amounts of computation. When solving a system exactly over the rational numbers, varying the right hand side can have a drastic effect on the bitsize of the solution and solve time.

In Section 2 we present background material in rational reconstruction and give some related results. In Section 3 we review Dixon's method and show how it is impacted by

---

[*] Research supported by NSF Grant CMMI-0726370 and ONR Grant N00014-08-1-1104.

applying output sensitive lifting. In Section 4 we introduce a new algorithm which integrates the iterative refinement procedure and rational reconstruction. Section 5 presents computational results and Section 6 contains our conclusions.

2 Rational Reconstruction

2.1 Background

Rational reconstruction is a necessary component of all the algorithms described in this paper. We briefly describe rational reconstruction and some related background material. The following well known result appears in [24] as Corollary 6.3a.

Theorem 2.1. *There exists a polynomial algorithm which, for a given rational number α and natural number B_d tests if there exists a rational number p/q with $1 \leq q \leq B_d$ and $|\alpha - p/q| < 1/(2B_d^2)$, and if so, finds this (unique) rational number.*

If an upper bound B_d is computed for the denominators of the components of x and a vector $\hat{x}$ satisfying $|\hat{x} - x|_\infty < 1/(2B_d^2)$ is computed, this theorem can be applied component-wise to $\hat{x}$ to compute the exact solution x. The following analogous result is given, in more generality, as Theorem 5.26 in [26].

Theorem 2.2. *There exists a polynomial algorithm which, for given natural numbers n, M, B_n, B_d, with $2B_nB_d \leq M$ tests if there exists a rational number p/q with $gcd(p,q) = 1$, $|p| < B_n$ and $1 \leq q < B_d$ such that $p = nq$ mod M, and if so, finds this (unique) rational number.*

Using this result a rational system of equations can be solved by scaling it to be integral, computing a solution to the system modulo an appropriate integer M and reconstructing the exact rational solution component-wise.

In both of the preceding theorems, the algorithms to reconstruct rational numbers rely on the Extended Euclidean Algorithm (EEA) to compute continued fraction convergents. The standard Euclidean Algorithm computes the greatest common divisor of integers m, n by repeatedly calculating the remainder of integer divisions. The EEA records additional information along the way, including the continued fraction expansion of m/n which is computed as a byproduct of the integer divisions. The continued fraction convergents provide a sequence of increasingly accurate rational approximations. They are best approximations in the sense that each convergent is closer to m/n than any number with smaller denominator. We use $[a_0; a_1, \ldots, a_k]$ to denote the continued fraction representation of a rational number r, and we will call the rational number $\frac{p_i}{q_i}$ representing $[a_0; a_1, \ldots, a_i]$ the i^{th} convergent of r.

Algorithm 1 Euclidean Algorithm

```
Input: integers $m, n$
$r_0 := n$,   $r_1 := m$,   $i := 1$
while $r_i \neq 0$ do
    $r_{i+1} := r_{i-1} \mod r_i$
    $i := i + 1$
end while
$l := i - 1$
Return: $r_l$=gcd($m, n$)
```

Algorithm 1 gives a description of the Euclidean Algorithm. The EEA will perform the same operations as the Euclidean Algorithm and its output will include the remainder sequence $r_0, \ldots, r_l$ in addition to the quotient sequence $a_0, \ldots, a_{l-1}$, where $a_i := \left\lfloor \frac{r_i}{r_{i+1}} \right\rfloor$ and the matrix sequence defined by:

$$Q_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad Q_i = Q_{i-1} \begin{pmatrix} a_{i-1} & 1 \\ 1 & 0 \end{pmatrix} \quad \forall i \geq 1.$$

There are several equivalent ways to define the matrix sequence and this notation is the most convenient for our purposes. We now state some basic results concerning continued fractions; these appear (with varying notation) in either Section 3.2 of [26] or Section 12.2 of [22].

Remark 2.3. *Suppose $r = m/n$ for integers m and $n \geq 1$, let r_i be the output of Algorithm 1 and a_i, Q_i be as defined above. Also let $\frac{p_i}{q_i}$ be the i^{th} convergent of r, and define $p_{i-2} = 0$, $q_{i-2} = 1$, $p_{i-1} = 1$, $q_{i-1} = 0$. Then the following relations hold:*

1. *For $k \geq 0$, $p_k = a_k p_{k-1} + p_{k-2}$ and $q_k = a_k q_{k-1} + q_{k-2}$.*
2. $\left| \frac{p_i}{q_i} - \frac{p_{i+1}}{q_{i+1}} \right| = \frac{1}{q_i q_{i+1}}$.
3. *If $r > 0$ then $\frac{p_1}{q_1} < \frac{p_3}{q_3} < \cdots < r < \cdots < \frac{p_4}{q_4} < \frac{p_2}{q_2}$.*
4. *If $r < 0$ then $\frac{p_2}{q_2} < \frac{p_4}{q_4} < \cdots < r < \cdots < \frac{p_3}{q_3} < \frac{p_1}{q_1}$.*
5. $Q_i = \begin{pmatrix} p_{i-1} & p_{i-2} \\ q_{i-1} & q_{i-2} \end{pmatrix} \quad \forall i \geq 0$
6. $\begin{pmatrix} m \\ n \end{pmatrix} = Q_i \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$.
7. $\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q_i^{-1} \begin{pmatrix} m \\ n \end{pmatrix} = (-1)^i \begin{pmatrix} q_{i-2} & -p_{i-2} \\ -q_{i-1} & p_{i-1} \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix}$.

A straightforward implementation of rational reconstruction will require $O(d^2)$ bit operations where d is the number of bits used to represent the input. In recent articles, such as [18, 20, 21], rational reconstruction algorithms that use $O(M(d)\log(d))$ bit operations where $M(d)$ is the cost of multiplication of integers with size bounded by 2^d. Using fast multiplication of [23] we have $M(d) = O(d \log(d) \log\log(d))$. This speedup of rational reconstruction is achieved using similar ideas to the fast Extended Euclidean Algorithm.

2.2 Certifying Solution Vectors

We now consider some sufficient conditions that can be efficiently checked to certify correctness of a rational system of equations. A lemma similar to the following was given by [3] and was used in [5, 6]. It can be used to certify correctness of reconstructed solutions while requiring little computation. Within this section we will use $\|A\|_{\max} = \max |a_{ij}|$.

Lemma 2.4. *Suppose A is a square integer matrix, y, b are integer vectors, and $d \geq 0$ is an integer. If for some integer M*

$$Ay = bd \mod M \qquad \textit{and}$$

$$\max(d\|b\|_\infty, n\|A\|_{\max}\|y\|_\infty) < M/2$$

then $Ay = bd$.

Proof. Suppose the conditions hold but $Ay - bd \neq 0$. Since $Ay - bd$ must be integral and $Ay = bd \mod M$ we have $\|Ay - bd\|_\infty \geq M$. But we also have

$$\begin{aligned}\|Ay - bd\|_\infty &\leq \|Ay\|_\infty + \|bd\|_\infty \\ &\leq 2\max(d\|b\|_\infty, n\|A\|_{\max}\|y\|_\infty) < M\end{aligned}$$

which gives a contradiction. □

We also make the observation that the statement of this lemma can be adjusted by replacing $n\|A\|_{\max}\|y\|_\infty$ with $\|A^T\|_2\|y\|_2$, and the proof will carry through identically by the Cauchy-Schwartz inequality.

Corollary 2.5. *Suppose A, b, y, d are as in Lemma 2.4. If $Ay = bd$ then $d \neq 0$ implies $x = y/d$ solves $Ax = b$, and $d = 0$ implies singularity of A.*

Suppose a solution to a system of equations is computed modulo p^k for some integer k and rational reconstruction is attempted without knowledge of valid bounds, by using guessed bounds such as $B_n = B_d = \lceil\sqrt{p^k/2}\rceil$ as in Theorem 2.2. In such a case, since B_n, B_d are not known to be valid, the reconstructed solution may be incorrect. Lemma 2.4 gives a very easily checked condition to certify correctness of the solution. If the solution is known to satisfy the modular system of equations, then checking the remaining conditions of the theorem requires only a few multiplications, in contrast to a high precision matrix-vector multiplication required to evaluate the linear equations exactly.

We now provide an analogue for the case when rational numbers are reconstructed from approximate solutions.

Lemma 2.6. *Suppose A is a square integer matrix, b is an integer vector and x is a rational vector that is known to satisfy $\|x - A^{-1}b\|_\infty < \epsilon$. If $x = y/d$, where y is an integer vector, and d is an integer satisfying $d < 1/(n\|A\|_{\max}\epsilon)$, then $Ax = b$.*

Proof. Suppose $\hat{x} = A^{-1}b$ and $Ax \neq b$. Since $Ay - bd \neq 0$ is integral, $\|Ay - bd\|_\infty \geq 1$. Next $d < 1/(n\|A\|_{\max}\epsilon)$ and $\|x - \hat{x}\|_\infty \leq \epsilon$ implies $\|x - \hat{x}\|_\infty < 1/(nd\|A\|_{\max})$. So we have

$$\begin{aligned}\|Ay - bd\|_\infty = d\|Ax - b\|_\infty &= d\|A(x - \hat{x})\|_\infty \\ &\leq dn\|A\|_{\max}\|x - \hat{x}\|_\infty < 1\end{aligned}$$

which gives a contradiction. □

The preceding lemma implies that if a rational solution x is reconstructed from an approximate solution, where the common denominator of the vector x is small enough, and its accuracy is known to satisfy a required bound, then its correctness can be certified without evaluating the equations.

While Lemmas 2.4 and 2.6 provide conditions to quickly certify correctness of solutions that have been reconstructed, their conditions are not necessary, and a correct rational solution may fail to satisfy them. The following example illustrates that this gap can depend on both the dimension and size of the data entries.

Example 2.7. *Suppose $A = aI_n$ for an integer a and I_n is the n dimensional identity matrix. For an n dimensional vector $b = [a, a, \ldots, a]^T$, $x = y/d = [1, 1, \ldots, 1]^T/1$ is a solution to $Ax = b$ for all positive integers a, n. After solving this system modulo $M \geq 2$ for a number M not dividing a, the correct solution will be reconstructed successfully. However, the conditions in Lemma 2.4 will not be met unless a solution is computed modulo $M \geq 2na$.*

This example also can be applied to Lemma 2.6, where we see that any value of $\epsilon \leq 1/2$ is sufficient for the correct solution to be determined using rational reconstruction, however the conditions are not satisfied unless the system is solved to within an error $\epsilon \leq 1/(2na)$.

Therefore, to design an algorithm that will compute and certify the correct rational solution as soon as possible, these techniques have both practical and theoretical drawbacks. We also mention that if an incorrect solution vector is checked for correctness by evaluating the linear equations of the system, its incorrectness can likely be discovered after evaluating only a small number of the equations. Therefore, although evaluating all of the linear equations could be computationally expensive, we expect identifying incorrectness of solutions to be considerably faster.

We now provide some necessary and sufficient conditions that can be used to verify correctness of a reconstructed solution. While these conditions are not as easily checked as those in the previously discussed results they can be easier to verify than evaluating the equations using full precision.

Lemma 2.8. *Suppose A is a square integer matrix, y, b are integer vectors, d is a positive integer and $x = y/d$. Then $Ax = b$ if and only if there exists an integer $M \geq 1$ such that*

$$Ay = bd \mod M \quad \textit{and} \quad \|Ax - b\|_\infty < M/d.$$

Proof. If $Ax = b$ then for any integer $M \geq 1$ the modular equation must hold and $\|Ax - b\|_\infty = 0 < M/d$. For the reverse direction suppose $Ax \neq b$, then for any positive integer M $Ay = bd \mod M$ implies $\|Ay - bd\|_\infty \geq M$, which means $\|Ax - b\|_\infty \geq M/d$ so both conditions can not hold at once. □

Thus, if solution y/d is known to satisfy $Ay = bd \mod M$, to check $Ax = b$ it is necessary and sufficient that $\|Ax - b\|_\infty < M/d$, which can be verified using approximations or truncated data. Similarly, if we have computed a rational solution $x = y/d$ satisfying $\|Ax - b\|_\infty < \epsilon$, this result tells us that instead of explicitly checking $Ax = b$, it is sufficient to select any positive integer $M \geq d/\epsilon$ and verify that $Ay = bd \mod M$. Evaluating this modular system will require less computation than verifying the full precision equations, especially when d/ϵ is reasonably small.

Related results appear in [13] who observes that under some assumptions, if the solutions at two (or more) consecutive lifting steps are the same, then there is a high probability that they give the correct answer. As our focus here is on deterministic algorithms we refer the reader to this reference for more information.

2.3 Warm Starting Rational Reconstruction

For the algorithm presented in Section 4 it is of interest to understand how the output of the Extended Euclidean Algorithm, and rational reconstruction, can change when its input is slightly perturbed. Understanding this will help us to perform warm starts for the rational reconstruction algorithm corresponding to Theorem 2.1. The following appears as Theorem A in [17]; related results appear in [20, 18].

Theorem 2.9. *Let $\frac{p_{k-1}}{q_{k-1}}, \frac{p_k}{q_k}$ be two consecutive convergents to a number β. Then these fractions are consecutive convergents to α if and only if*

$$\left|\alpha - \frac{p_k}{q_k}\right| < \frac{1}{q_k(q_k + q_{k-1})}.$$

This theorem gives conditions which can be used to verify that a sequence of continued fraction approximations is correct up to a certain point. The following result applies this theorem to the framework of rational reconstruction.

Theorem 2.10. *Let* x, α *be a rational numbers satisfying* $|x-\alpha| < 1/(2B^2)$ *for some integer* B. *Suppose* $\frac{p_k}{q_k}$ *is any continued fraction convergent of* x *such that* $q_k < B$. *If* $k \geq 3$ *then either* $\frac{p_{k-2}}{q_{k-2}}, \frac{p_{k-1}}{q_{k-1}}$ *or* $\frac{p_{k-1}}{q_{k-1}}, \frac{p_k}{q_k}$ *are two consecutive convergents of* α.

Proof. Without loss of generality we may assume $\frac{p_{k-1}}{q_{k-1}} \leq x \leq \frac{p_k}{q_k}$. First suppose $|\alpha - \frac{p_{k-1}}{q_{k-1}}| < \frac{1}{q_k q_{k-1}}$. Then by Remark 2.3 if $k \geq 1$, $q_k \geq q_{k-1} + q_{k-2}$, so we have

$$\left|\alpha - \frac{p_{k-1}}{q_{k-1}}\right| < \frac{1}{q_k q_{k-1}} \leq \frac{1}{q_{k-1}(q_{k-1}+q_{k-2})}$$

and by Theorem 2.9, $\frac{p_{k-2}}{q_{k-2}}, \frac{p_{k-1}}{q_{k-1}}$ are two consecutive convergents of α. So we may assume that $|\alpha - \frac{p_{k-1}}{q_{k-1}}| \geq \frac{1}{q_k q_{k-1}}$. From $|\frac{p_k}{q_k} - \frac{p_{k-1}}{q_{k-1}}| = \frac{1}{q_k q_{k-1}}$ and $\frac{p_{k-1}}{q_{k-1}} \leq x$ it follows that $\frac{p_k}{q_k} \leq \alpha$. Finally $|x-\alpha| < \frac{1}{2B^2}$ and $x \leq \frac{p_k}{q_k}$ gives

$$\left|\alpha - \frac{p_k}{q_k}\right| < \frac{1}{2B^2} \leq \frac{1}{2q_k^2} \leq \frac{1}{q_k(q_k+q_{k-1})}.$$

By Theorem 2.9 we have $\frac{p_{k-1}}{q_{k-1}}, \frac{p_k}{q_k}$ as two consecutive convergents of α, which establishes our desired result. □

Thus, if rational reconstruction is performed using an approximate input $x \approx \alpha$, the intermediate steps of the EEA will be correct in all but possibly the last step. If x is later refined to a more accurate approximation of α then in order to apply rational reconstruction again, we can start the algorithm where it left off, with the need for at most one step backward.

3 Output Sensitive Lifting

Algorithm 2 describes Dixon's well known algorithm for solving a integer system of equations $Ax = b$ [8]. His algorithm has three steps; first an inverse of $A \mod p$ is computed, second p-adic lifting is used to find a solution mod p^k, then the rational solution is reconstructed. Dixon showed that if $Ax = b$ is an n by n integer system of equations and $\|A\|_{\max}, \|b\|_\infty$ are bounded by a constant, then the algorithm terminates with at most $O(n^3\log^2(n))$ bit operations. We will review how that figure was obtained. For simplicity it is assumed the entries of A and b are bounded by a constant. He first finds a word sized prime p not dividing $det(A)$. The inversion of A mod p will require $O(n^3)$ operations. There will be $O(\log(B))$ lifting steps. In each lifting step, the computational cost is dominated by computing d, which has integral entries that have bitsize at most $O(\log(n))$. Thus each lifting step will cost $O(n^2\log(n))$ giving a total cost of $O(n^2\log(n)\log(B))$ for all the lifting steps. The final rational reconstruction, using the Extended Euclidean Algorithm component-wise will cost $O(n\log^2(B))$, (or faster using recent accelerated rational reconstruction techniques). We also have $\log(B) = \log(2\|A\|_2^{2n-1}\|b\|_2) = O(n\log(n))$, so the bit complexity is

$$O(n^3 + n^2\log(n)\log(B) + n\log^2(B)) = O(n^3\log^2 n).$$

Algorithm 2 Standard Dixon Algorithm

```
Input: $A, b$                    {$Ax = b$ is system to be solved}
Determine $p$ not dividing $det(A)$
Compute $A^{-1}$ mod $p$
$\hat{x} := 0, i := 0, d := b, B := 2\|A\|_2^{2n-1}\|b\|_2$
while $p^i < B$ do
    $y := A^{-1}d \mod p$
    $\hat{x} := \hat{x} + yp^i$        {This will set $\hat{x} = A^{-1}b \mod p^{i+1}$ }
    $d := \left(\frac{d-Ay}{p}\right)$
    $i := i + 1$
end while
$x$:=Reconstruct($\hat{x}, p^i$)
Return: $x$                                  {Solution to system}
```

Algorithm 3 describes the output sensitive version of Dixon's Algorithm. In this variant of the algorithm, instead of waiting until the modulus of the intermediate solution $\hat{x}$ exceeds the bound B, rational reconstruction is attempted at intermediate steps. Success of the rational reconstruction is not theoretically guaranteed at these steps, so the reconstructed solution must be verified. By Theorem 2.2, the reconstruction routine is guaranteed to succeed when p^{i+1} exceeds the (unknown) quantity $2S$ where $\log(S) = bitsize(A^{-1}b)$. Using this notation, S represents the largest value which is the product of a numerator and denominator of a component of the solution vector, and we have $2S \leq B = 2\|A\|_2^{2n-1}\|b\|_2$. The bit complexity of the output sensitive Dixon algorithm will depend on which steps are specified as reconstruction steps. Here we will make the choice that reconstruction is attempted at steps for which $i = 2^k$ for some integer k. This choice of frequency is important. For example, if reconstruction is attempted at predetermined constant length intervals the bit complexity of the algorithm would change. For simplicity of the analysis we will assume we are working with systems whose entries are bounded by a constant.

Algorithm 3 Output Sensitive Dixon Algorithm

```
Input: $A, b$                    {$Ax = b$ is system to be solved}
Determine $p$ not dividing $det(A)$
Compute $A^{-1}$ mod $p$
$\hat{x} := 0, i := 0, d := b$
while solution not found do
    $y := A^{-1}d \mod p$
    $\hat{x} := \hat{x} + yp^i$
    if reconstruction step then
        $x$:=Reconstruct($\hat{x}, p^{i+1}$)
        Check $Ax = b$
    end if
    $d := \left(\frac{d-Ay}{p}\right)$
    $i := i + 1$
end while
Return: $x$                                  {Solution to system}
```

Theorem 3.1. *Let* $Ax = b$ *be a square nonsingular integer system of equations. If the entries of* A, b *are bounded by a constant and* $\log(S) = bitsize(A^{-1}b)$ *then the Output Sensitive Dixon Algorithm terminates with* $O(n^3 + n^2\log(n)\log(S))$ *bit operations.*

Proof. Determining p and computing $A \bmod p$ will cost $O(n^3)$ operations. The number of loops will be $O(\log(S))$ because our choice of reconstruction frequency ensures we will perform at most twice the necessary number of loops. The number of operations performed in each loop, excluding the cost of the rational reconstruction attempts, is $O(n^2\log(n))$ as in the standard Dixon Algorithm. The computational cost of performing rational reconstruction while in loop i is $O(ni^2)$. The cost of checking the solution in loop i will be $O(n^2 i)$, since it requires performing a matrix-vector multiplication where the entries of the matrix are bounded by a constant, and the entries of the vector are bounded by $2^{O(i)}$. Thus, since reconstruction will be attempted and verified at steps $i = 2^k$ for $k = 1, 2, \ldots, O(\log\log(S))$ we have the following bound on the total combined cost of rational reconstruction and checking the solutions:

$$\sum_{k=1}^{O(\log\log(S))} \Big(O(n(2^k)^2) + O(n^2 2^k)\Big)$$

$$= O(n\log^2(S) + n^2\log(S)).$$

Using $\log(S) = O(n\log(n))$ we have the following bound on the number of bit operations $O(n^3 + n^2\log(n)\log(S) + n\log^2(S) + n^2\log(S)) = O(n^3 + n^2\log(n)\log(S))$, which is our desired result. $\square$

We see that even if $S = \Omega(B)$, this matches the worst case complexity for the standard Dixon algorithm. If $\log(S)$ is bounded by a constant, or is $O(n/\log(n))$, the bit complexity becomes $O(n^3)$ which is the best we could hope for.

4 An Integrated Approach

When used to find exact solutions to linear equations, rational reconstruction is typically used as a separable routine, called after or within an iterative lifting procedure. We show a procedure to directly integrate rational reconstruction into an iterative refinement procedure. Although this will not improve the bit complexity of the output sensitive algorithm, it is theoretically interesting and reduces the number of operations.

The iterative refinement procedure for solving linear systems of equations calculates a highly accurate approximate solution in order to apply Theorem 2.1 to reconstruct the rational solution. The solution is first calculated approximately using numerical methods, then the error is computed exactly and iteratively corrected. In order to guarantee correctness of the reconstructed solution, Cramer's rule and the Hadamard bound are used, as in Dixon's method, to bound the denominators of the solutions. The iterative structure of this algorithm is similar to Dixon's method and rational reconstruction can also be attempted at intermediate steps to make it output sensitive. Algorithm 4 gives a description of the iterative refinement algorithm similar to the algorithm of Wan [27]. This algorithm does require the assumptions that the final solution can be approximately represented as a floating point number, and that the matrix can be successfully numerically factored or inverted.

By Theorem 2.10, if rational reconstruction is attempted component-wise at an intermediate step of the lifting algorithm, then many steps of the EEA performed will be correct, even if the reconstructed solution is not correct. Thus, if rational reconstruction is performed at an intermediate step and it fails to find the correct rational solution, the intermediate steps of the EEA calculated can be stored and used to warm start rational reconstruction in a later loop. The following lemma gives an explicit formula for updating the intermediate work of the EEA when a solution is refined.

Algorithm 4 Iterative Refinement Method

```
Input: $A, b$                    {$Ax = b$ is system to be solved}
Compute numerical LU factorization of A
$N := 0$                         {Numerator of the approximation}
$D := 1$                         {Common denominator of approximation}
$B := 2\|A\|_2^{2n}$
$\Delta := b$                    {Error measure of solution at each step}
while $D < B$ do
    Compute $\hat{x} :\approx A^{-1}\Delta$    {Using numerical LU factorization}
    Choose an integer $\alpha < \frac{||\Delta||_\infty}{||\Delta - A\hat{x}||_\infty}$ where $\alpha\hat{x}$ is within floating point range
    Set $[\hat{x}] \approx \alpha\hat{x}$        {Round to the nearest integer}
    $\Delta := \alpha\Delta - A[\hat{x}]$        {Update the residual}
    $D := D \times \alpha$                       {Update the denominator}
    $N := N \times \alpha + [\hat{x}]$
end while
Reconstruct $x$ using $N/D$
Return: $x$                      {Solution to system}
```

Lemma 4.1. *Let $\hat{m}/\hat{n}$ be a rational number and suppose at the i^{th} step of the EEA the following have been computed*

$$\hat{Q}_i = \begin{pmatrix} \hat{p}_{i-1} & \hat{p}_{i-2} \\ \hat{q}_{i-1} & \hat{q}_{i-2} \end{pmatrix}, \quad \hat{r}_i, \quad \hat{r}_{i+1}.$$

Let m/n be a rational number for which $Q_i = \hat{Q}_i$ is known to be a matrix encountered in the application of the EEA. Then if $\frac{m}{n} = \frac{\hat{m}\alpha+\Delta}{\hat{n}\alpha}$, the following relation gives the values of r_i, r_{i+1}, the remainders encountered at the i^{th} step of the EEA when applied to m, n:

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \alpha \begin{pmatrix} \hat{r}_i \\ \hat{r}_{i+1} \end{pmatrix} + (-1)^i \Delta \begin{pmatrix} \hat{q}_{i-2} \\ -\hat{q}_{i-1} \end{pmatrix}.$$

Proof. By Remark 2.3 we have the following

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = Q_i^{-1}\begin{pmatrix} m \\ n \end{pmatrix} = \alpha Q_i^{-1}\begin{pmatrix} \hat{m} \\ \hat{n} \end{pmatrix} + \Delta Q_i^{-1}\begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= \alpha \begin{pmatrix} \hat{r}_i \\ \hat{r}_{i+1} \end{pmatrix} + (-1)^i \Delta \begin{pmatrix} \hat{q}_{i-2} \\ -\hat{q}_{i-1} \end{pmatrix}$$

which gives our proposed formula. $\square$

This gives us a nice way to update the remainder sequence, r_i, r_{i+1}, from $\hat{r}_i, \hat{r}_{i+1}$ without requiring to access m or n. We need only know a scale factor α and difference Δ, which might have a much smaller representation than m or n.

Algorithm 5 gives a description of the iterative refinement with integrated rational reconstruction. The basic structure of this algorithm follows the methods of [27], but has been modified to incorporate rational reconstruction within the routine.

Observe that the numerator of the current approximate solution which was stored as N in Algorithm 4 is no longer

Algorithm 5 Integrated Iterative Refinement Method

```
Input: $A, b$    {$Ax = b$ is system to be solved}
Compute numerical LU factorization of A
$D := 1$    {Common denominator of approximation}
$(Q_0^i, r_0^i, r_1^i) = (I_2, 0, 1) \quad \forall i \in 1, \ldots, \dim(A)$    {Data for EEA for each component}
$\Delta := b$    {Error measure of solution at each step}
while solution not found do
    Compute $\hat{x} :\approx A^{-1}\Delta$    {Using numerical LU factorization}
    Choose an integer $\alpha < \frac{||\Delta||_\infty}{||\Delta - A\hat{x}||_\infty}$ where $\alpha\hat{x}$ is within floating point range
    Set $[\hat{x}] \approx \alpha\hat{x}$    {Round to the nearest integer}
    $\Delta := \alpha\Delta - A[\hat{x}]$    {Update the residual}
    $D := D \times \alpha$    {Update the common denominator of approximation}
    Update $Q_k^i, r_k^i, r_{k+1}^i \forall i$ using $[\hat{x}]_i, \alpha$ and Lemma 4.1
    Perform additional steps of EEA on $(Q_k^i, r_k^i, r_{k+1}^i)$ maintaining $q_{k-1}^i < \sqrt{2D}$
    {The intermediate reconstructed solution $x$ is defined by, $x_i = p_{k-1}^i/q_{k-1}^i$.}
    Check $Ax = b$    {Use full precision check if step in loop is a power of 2, otherwise use the quick check of Lemma 2.6}
end while
Return: $x$    {Solution to system}
```

explicitly stored, but is implicitly represented by the partially reconstructed solution components and remainders which are updated at each step. At each step we know that the approximation of the i^{th} solution component which was represented by N_i/D in Algorithm 4 is stored implicitly as

$$\begin{pmatrix} N_i \\ D \end{pmatrix} = Q_k^i \begin{pmatrix} r_k^i \\ r_{k+1}^i \end{pmatrix}$$

in Algorithm 5. Since the intermediate reconstructed solutions are always available and updated, it makes sense to apply the quick correctness check of Lemma 2.6 frequently. We still apply the expensive full precision system check, but with the same frequency as in Algorithm 3 so we do not increase the worst case complexity but still guarantee the algorithm terminates after $O(log(S))$ loops.

Remark 4.2. *If A is square matrix which can successfully be numerically factored, the entries of A, b and $A^{-1}b$ are bounded by a constant and $\log(S) = bitsize(A^{-1}b)$ then the integrated iterative-refinement method as described in Algorithm 5 will terminate with the correct solution to $Ax = b$ after performing $O(n^3 + n^2 \log(n) \log(S))$ bit operations.*

The structure of the algorithm here mirrors the Output Sensitive Dixon's method which was analyzed in the previous section, so we only note the differences here. The non-output sensitive version of the iterative-refinement method is shown to be correct and analyzed in [27]. The only significant difference here is the rational reconstruction, and by Theorem 2.10 we will perform asymptotically the same amount of computation as if rational reconstruction had been applied a single time to each component at the time of termination. Doing the additional checks if $Ax = b$ using Lemma 2.6 at every loop of the refinement procedure can be done in constant time in each loop. We note that for p-adic lifting, this approach can not be applied in the same way. Iterative refinement computes an approximate solution in a top down fashion, with each refinement making smaller and smaller adjustments leaving the leading digits unchanged. For p-adic lifting, the solution is computed from the bottom up, and the leading digits of the modular solution change at every iteration.

5 Computational Results

In this section we present computational results to compare the performance of the methods described in this paper. Source codes for the methods tested here and scripts to generate the test problems are freely available at

`www.isye.gatech.edu/~dsteffy/rational/`

for any research purposes.

5.1 Implementation

Output sensitive lifting can be applied in both the sparse and dense case. It can be applied in both the modular (i.e. Dixon) or numerical (iterative refinement) settings. For our computations we have chosen to evaluate it in the dense setting using both a Dixon based solver and an iterative refinement based solver. The reason we have chosen to consider both these methods is that the Dixon based solver can be tested on some well known problems which are too ill-conditioned for a numerical solver to handle, and testing the iterative refinement solver allows us to evaluate the integrated iterative-refinement method in comparison with the standard and output sensitive methods. Moreover, in [7] output sensitive lifting was tested within many methods in the sparse setting and found it to be highly successful for a large class of applied problems.

In all of the implementations we have produced code that works for dense rational systems of equations. The rational inputs are scaled to be integer before applying the algorithms. We use a standard implementation of rational reconstruction, employing a technique referred to as the DLCM method in Section 3.2.3 of [7]. This technique amounts to storing the LCM of the denominators of the reconstructed solution vector and using this information to accelerate component-wise reconstruction by fixing factors of the component denominators or terminating early if this LCM grows too large.

Our implementation of Dixon's algorithm is written in C/C++ and uses the FFLAS and FFPACK packages [11], which provide fast BLAS and LAPACK routines for finite fields in C++. We implemented both the standard Dixon algorithm as described in Algorithm 2 along with the output sensitive Dixon algorithm as given in Algorithm 3.

The implementation of the iterative-refinement methods is written in C and uses the BLAS [9, 16] and LAPACK [2] routines for the dense numerical linear algebra. We have used the ATLAS package [28] which provides automatically tuned BLAS and a subset of LAPACK routines. We implemented three strategies for rational reconstruction for the iterative-refinement method. First, we use the Hadamard bound as in Algorithm 4; second, we attempt reconstruction at loops which are a power of two, analogous to Algorithm 3; third, we implement a version of Algorithm 5 where the partially reconstructed solutions are stored from step to step.

We also comment that the purpose of our implementations were to accurately compare ideas described in this article in a straightforward implementation. The implementations are not expected to be competitive with state of the art solvers such as LinBox [10].

5.2 Test Problems

The goal of our computational experiments is two fold. We seek to evaluate how output sensitive lifting can accelerate linear system solving on problems for which the bitsize of the final solution is small, problems where it should have a distinct advantage. Additionally, we seek to compare the speed of the standard and output sensitive algorithms on problems whose output is very large, to verify that in the worst case there is no significant drawback to using output sensitive lifting.

In order to meet these goals and adequately compare the algorithms we chose a variety of problems in our test set. Table 1 provides descriptions of the classes of dense matrices which we use to test our methods.

Table 1: Description of test matrices

Matrix Type	Construction
Hadamard D_n for $n = 2^k$	$D_1 = (1)$ and $D_n = \begin{pmatrix} D_{n/2} & D_{n/2} \\ D_{n/2} & -D_{n/2} \end{pmatrix}$
Random R_n	$\{R_n\}_{ij} \in [-100, 100]$ if $i \neq j$, and $\{R_n\}_{ii} = 10,000$
Hilbert H_n	$\{H_n\}_{ij} = 1/(i + j - 1)$
Vandermonde V_n	$\{V_n\}_{ij} = i^{j-1}$
Lehmer L_n	$\{L_n\}_{ij} = \min(i, j)/\max(i, j)$

We mention that there was some difficulty in choosing which problems to look at. It is difficult to find an explicit linear system of equations for which the bitsize of the solution meets the Hadamard bound exactly. The Hadamard matrices have determinants which meet the Hadamard determinant bound tightly. However, the solution bitsize will not meet this bound, because although $det(D_n) = 2^{n-1}$, for any right hand side vector with bounded bitsize the solution will not have bitsize $\Omega(n)$ because $D_n^{-1} = \frac{1}{n} D_n$.

We also consider randomly generated dense matrices. For these matrices we choose the entries uniformly at random from integers with absolute value at most 100, and assign the diagonal entries all to 10,000 to ensure numerical stability. The Hilbert matrix is a famous and often cited example of an ill-conditioned matrix, and it is impossibly difficult for numerical solvers to tackle, even at low dimension. We use a type of Vandermonde matrix with the rows generated by increasing integers as described in the table. The Lehmer matrices are also a well known class of ill-conditioned matrices.

Choosing right hand sides for the systems of equations is also an important consideration. Some computational linear algebra studies use arbitrary right hand sides, such as setting b equal to the sum of the columns in A, giving a solution of all ones. While in the numerical setting, this is a perfectly reasonable right hand side to consider, it is not appropriate in our case because the algorithms studied here have run times depending on the bitsize of the solutions. For our evaluations we will use the unit vector e_1 as the right hand side for each system. This is a reasonable choice because it corresponds to computing the first row of the inverse matrix, which should be adequately representative of the typical solution complexity.

5.3 Computations

Computations were performed on a linux machine with a 2.4 GHz AMD Opteron 250 processor and 4GB of RAM. Table 2 compares the standard Dixon algorithm and the output sensitive Dixon algorithm on the entire problem set; the solve times are given in seconds. The table also includes the log of the Hadamard bound on the solution bitsize $\log(B) = \log(2\|A\|_2^{2n-1}\|b\|_2)$, along with the actual value $\log(S) = bitsize(A^{-1}b) \leq \log(B) - 1$. In addition to the total solution time for each method we include a profile of how the time was spent in different stages of the algorithm. We divide the solution time between the following three tasks: the finite field matrix factorization, the p-adic lifting steps, and the rational reconstruction.

The first observation we make from the table is that in all problem classes, other than the randomly generated matrices, the Hadamard bound was a very weak upper bound on the solution size. On the set of randomly generated matrices, the Hadamard bound does provide a fairly tight bound on the final solution bitsize, and it is on these problems where we might expect the standard Dixon algorithm to have an advantage over the output sensitive Dixon. We see that even on these problems, the output sensitive Dixon algorithm has approximately the same solution times. This implies that even if the Hadamard bound is nearly tight our strategy for output sensitive lifting only performs a negligible amount of additional computation.

For the remainder of the problem set we see the output sensitive method having an advantage of several orders of magnitude. For example, we are able to compute the first row of the inverse of the dimension 2000 Hilbert matrix in just over 20 minutes. For some of these special systems, the representation of the matrices, and the representation of the scaled integer matrices, becomes very large and can be a limiting factor in the computation. In order to perform such special purpose computation the programs could be modified to perform the necessary calculations with out storing the matrix explicitly.

When testing the iterative refinement solver the Hilbert, Vandermonde and Lehmer matrices are were too numerically difficult for the numerical LAPACK solver to handle. We found that the integrated method described in section 4 did not make the solver any faster than using the basic output sensitive lifting strategy. Comparing the versions of the iterative refinement method using and not using output sensitive lifting led to very similar solve times and timing profile breakdowns as Dixon's algorithm so we omit these results. We also note that the portion of the time the iterative refinement spent on rational reconstruction was similar to that of the Dixon solver, so at least for this test set the actual time spent on rational reconstruction was not a bottleneck and thus any speedups would not be expected to improve the speed.

6 Conclusion

We have shown how output sensitive lifting can improve algorithms for symbolically solving systems of linear equations. Output sensitive algorithms allow for systems of rational linear equations to be solved very quickly when the

Table 2: Solve times for Dixon algorithms in seconds

Problem Details			Standard Dixon				Output Sensitive Dixon			
Matrix	$\log(B)$	$\log(S)$	**Total**	Factor	Lift	R.R.	**Total**	Factor	Lifting	R.R.
D_{1024}	12284	10	**30.07**	0.46	29.03	0.58	**1.10**	0.45	0.07	0.58
D_{2048}	24572	11	**231.09**	2.81	225.90	2.39	**5.37**	2.74	0.29	2.33
D_{4096}	57339	12	**2211.46**	18.86	2183.30	9.30	**28.98**	18.25	1.22	9.51
R_{200}	5588	5297	**3.08**	0.02	0.71	2.34	**3.08**	0.02	0.70	2.35
R_{500}	13988	13267	**57.99**	0.18	9.84	47.97	**57.81**	0.18	9.65	47.98
R_{1000}	27988	26555	**611.11**	0.87	73.92	536.31	**610.11**	0.87	73.23	536.02
H_{500}	1432292	1269	**5221.53**	0.18	5220.76	0.59	**6.45**	0.18	5.95	0.32
H_{1000}	5742567	2540	**188255.63**	0.86	188249.50	5.27	**85.32**	0.86	82.54	1.92
H_{2000}	22997129	5084	**-**	-	-	-	**1224.31**	4.34	1202.31	17.65
V_{100}	130944	793	**13.53**	0.00	13.40	0.12	**0.16**	0.00	0.07	0.09
V_{300}	1474141	3205	**4145.62**	0.06	4138.51	7.04	**8.54**	0.06	4.43	4.05
V_{500}	4469528	5948	**71704.65**	0.18	71654.01	50.45	**69.46**	0.18	44.73	24.55
L_{500}	727997	3	**280.74**	0.18	280.41	0.15	**0.31**	0.18	0.01	0.12
L_{1000}	2885996	3	**3493.41**	0.87	3491.85	0.70	**1.40**	0.87	0.04	0.50
L_{2000}	11531996	3	**56750.15**	4.39	56742.39	3.37	**6.49**	4.36	0.13	2.00

final solutions are small in size, while maintaining the same worst case bit complexity even when solutions are large in size. Our tests were performed on several types of dense systems with a variety of characteristics.

We introduced a way to warm start the rational reconstruction portion of the iterative refinement method. While this did not give a performance increase over the output sensitive version of this algorithm there may be other settings in which warm starting the EEA or rational reconstruction could prove helpful.

Throughout this paper we have primarily focused on output sensitive lifting applied to dense systems of equations; this technique is also fully applicable in the sparse setting. Output sensitive lifting an important tool for solving systems of linear equations exactly. Our computations suggest that any exact precision solver relying on iterative methods should employ output sensitive lifting.

References

[1] J. Abbott and T. Mulders. How tight is Hadamard's bound? *Experiment. Math.*, 10(3):331–336, 2001.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[3] S. Cabay. Exact solution of linear equations. In *SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 392–398, New York, NY, USA, 1971. ACM.

[4] L. Chen and M. Monagan. Algorithms for solving linear systems over cyclotomic fields. *Submitted*, 2008.

[5] Z. Chen. A BLAS based C library for exact linear algebra over integer matrices. Master's thesis, School of Computer Science, University of Waterloo, 2005.

[6] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In *ISSAC '05: Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation*, pages 92–99, New York, NY, USA, 2005. ACM.

[7] W. J. Cook and D. E. Steffy. Solving very sparse rational systems of equations. *Submitted*, 2009.

[8] J. D. Dixon. Exact solution of linear equations using p-adic expansion. *Numerische Mathematik*, 40:137–141, 1982.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[10] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. LinBox: A generic library for exact linear algebra. In A. M. Cohen, X.-S. Gao, and N. Takayama, editors, *Mathematical Software: ICMS 2002*, pages 40–50, Singapore, 2002. World Scientific.

[11] J.-G. Dumas, P. Giorgi, and C. Pernet. Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages. *ACM Transactions on Mathematical Software*, 35(3):Article 19, 2008.

[12] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Solving sparse rational linear systems. In *ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 63–70, New York, NY, USA, 2006. ACM.

[13] I. Z. Emiris. A complete implementation for computing general dimensional convex hulls. *International Journal of Computational Geometry and Applications*, 8:223–253, 1998.

[14] E. Kaltofen. An output-sensitive variant of the baby steps/giant steps determinant algorithm. In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 138–144, New York, NY, USA, 2002. ACM.

[15] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Proceedings of the Ninth International Symposium on Applied, Algebraic Algorithms, Error-Correcting Codes, Lecture Notes in Computer Science 539*, pages 29–38, Heidelberg, Germany, 1991. Springer.

[16] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[17] D. H. Lehmer. Euclid's algorithm for large numbers. *The American Mathematical Monthly*, 45(4):227–233, 1938.

[18] D. Lichtblau. Half-gcd and fast rational recovery. In *ISSAC '05: Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 231–236, New York, NY, USA, 2005. ACM.

[19] T. Mulders and A. Storjohann. Diophantine linear system solving. In *ISSAC '99: Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, pages 181–188, New York, NY, USA, 1999. ACM.

[20] V. Y. Pan and X. Wang. Acceleration of Euclidean algorithm and rational number reconstruction. *SIAM Journal of Computing*, 32:548–556, 2003.

[21] V. Y. Pan and X. Wang. On rational number reconstruction and approximation. *SIAM Journal of Computing*, 33:502–503, 2004.

[22] K. Rosen. *Elementary Number Theory.* Addison Wesley Longman, New York, 2000.

[23] A. Schonhage. Schnelle berechnung von kettenbruchentwicklungen. *Acta Inform.*, 1:139–144, 1971.

[24] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester, UK, 1986.

[25] S. Ursic and C. Patarra. Exact solution of systems of linear equations with iterative methods. *SIAM Journal on Matrix Analysis and Applications*, 4(1):111–115, 1983.

[26] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, Cambridge, UK, 2003.

[27] Z. Wan. An algorithm to solve integer linear systems exactly using numerical methods. *Journal of Symbolic Computation*, 41:621–632, 2006.

[28] R. C. Whaley and A. Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, 2005.

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