Exact Solutions to Linear Systems of Equations using Output Sensitive Lifting

Daniel E. 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 number 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 which allows the possibility of early termination when the solution size is small. 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 variant of the iterative-refinement method that incorporates warm starts into the rational reconstruction procedure.

1 Introduction

Solving rational or integer linear systems of equations is a well developed area of symbolic computation. Dixon [9] 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 [16, 36]. 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 [31, 33]. Others have further developed and analyzed these methods [6, 7, 11, 13, 22, 23].

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 $bitsize(p/q) = \lceil \log(|pq|) \rceil$. For a rational vector v we will define $size(v) = \max_i bitsize(n_i/d)$ where $n/d = v$ is a representation of v using an integer vector n and a common denominator d. 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 the final solution size. Similarly, if a rational solution is to be reconstructed from an approximate solution, the level of approximation depends on the size of the final solution. 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 vector has size bounded by $\log(\|A\|_2^{2n-1}\|b\|_2)$. 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 with the possibility of identifying the solution early and avoiding unnecessary lifting/refinement loops. The term output sensitive lifting is used by Chen and Storjohann [6, 7] 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 [15] where it is referred to as *early termination*. Output sensitive lifting was also studied in [5] 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 [8] 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 size 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 ratio-

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

nal numbers, varying the right hand side can have a drastic effect on the size of the solution and solve time.

This paper studies output sensitive techniques applied to two related classes of algorithms for solving linear systems. The first class of algorithm we consider is the p-adic lifting based strategy of Dixon [9] and the second algorithm is the iterative-refinement method developed by Wan [33]. Both algorithms have an iterative structure and are later defined as Algorithms 2 and 4. We will use the terminology of *p-adic lifting* and *lifting* when describing and referencing the Dixon algorithm because it constructs a modular p-adic solution from the *bottom up* in order to determine a rational solution. We will use the terminology of *iterative refinement* or *refinement* to describe Wan's Algorithm because it is based on iteratively refining an approximate solution, constructing an approximate solution in a *top down* manner. The similar structure of the Algorithms of Dixon and Wan allows output sensitive lifting to be applied in a similar way in both cases.

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 applying output sensitive lifting. In Section 4 we describe two output sensitive versions of the iterative-refinement method, one of which incorporates warm starts for 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 [28] 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. Theorem 2.1 is used for this purpose in the iterative-refinement method later described as Algorithm 4.

The following result is given, in more generality, as Theorem 5.26 in [32] and is analogous to Theorem 2.1.

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 $p = nq \bmod M$, 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. Theorem 2.2 is used for this purpose in Dixon's Algorithm which is later described as Algorithm 2.

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 m/n, and we will call the rational number $\frac{p_i}{q_i}$ representing $[a_0; a_1, \ldots, a_i]$ the ith convergent of m/n.

Algorithm 1 Euclidean Algorithm
```
Input: integers $m, n$
$r_0 := n,\quad r_1 := m,\quad i := 1$
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 [32] or Section 12.2 of [26].

Remark 2.3. *Consider the rational number 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 ith 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 $k \geq 0$, $p_k = a_kp_{k-1} + p_{k-2}$ and $q_k = a_kq_{k-1} + q_{k-2}$.*
2. $\left| \frac{p_i}{q_i} - \frac{p_{i+1}}{q_{i+1}} \right| = \frac{1}{q_iq_{i+1}}$.
3. *If* $\frac{m}{n} > 0$ *then* $\frac{p_1}{q_1} < \frac{p_3}{q_3} < \cdots < \frac{m}{n} < \cdots < \frac{p_4}{q_4} < \frac{p_2}{q_2}$.
4. *If* $\frac{m}{n} < 0$ *then* $\frac{p_2}{q_2} < \frac{p_4}{q_4} < \cdots < \frac{m}{n} < \cdots < \frac{p_3}{q_3} < \frac{p_1}{q_1}$.
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. Recent articles including [20, 24, 25] describe 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 [27] 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 [4] and was used in [6, 7]. It can be used to certify correctness of reconstructed solutions while requiring little computation. Throughout the rest of the paper 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

$$\|Ay - bd\|_\infty \leq \|Ay\|_\infty + \|bd\|_\infty$$

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

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 satisfy the conditions of 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

$$\|Ay - bd\|_\infty = d\|Ax - b\|_\infty = d\|A(x - \hat{x})\|_\infty$$

$$\leq dn\|A\|_{\max}\|x - \hat{x}\|_\infty < 1$$

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 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 both of the following conditions hold.*

$$Ay = bd \mod M \tag{1}$$

$$\|Ax - b\|_\infty < M/d \tag{2}$$

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 interval arithmetic. 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 also appear in [14] where it is shown, under some assumptions, that if the solutions at two (or more) consecutive lifting steps are the same there is a high probability that they give the correct answer.

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 [19]; related results appear in [20, 24].

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 for Dixon's Method

Algorithm 2 describes Dixon's well known algorithm for solving an integer system of equations $Ax = b$ [9]. This algorithm is sometimes referred to as the *p-adic lifting algorithm* for solving linear systems and related ideas are considered by other authors [17, 21, 34].

Algorithm 2 Standard Dixon Algorithm

```
Input: $A, b, p$        {$Ax = b$ is system to be solved, $p$ is a prime 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}
```

His algorithm has three steps; first an inverse of $A \mod p$ is computed, second p-adic lifting is used to construct a solution mod p^k, then the rational solution is reconstructed. Dixon showed the following bound regarding the complexity of his algorithm assuming $Ax = b$ is an n dimensional square system of equations and $\|A\|_{\max}, \|b\|_\infty$ are bounded by a constant. In his analysis he also assumed that a prime p not dividing $det(A)$ and bounded by a constant not depending on A was found (such a prime might not exist).

Theorem 3.1. *Let $Ax = b$ be an n dimensional square nonsingular integer system of equations. If the entries of A, b are bounded by a constant and p is bounded by a constant then Algorithm 2 will find the rational solution using $O(n^3 \log^2(n))$ bit operations.*

We will review how this bound was obtained. The inversion of A mod p can be done with $O(n^3)$ operations. There will be $O(\log(B))$ lifting steps. At the i^{th} iteration of the algorithm, entries of $\hat{x}$ will be in the range $[0, p^{i+1} - 1]$ and d is updated to equal $(b - A\hat{x})/p^i$. Therefore d will have integral entries with bitsize $O(\log n)$. Updating $y, \hat{x}$ and d in each lifting step is accomplished with $O(n^2 \log(n))$ bit operations. This gives a total cost of $O(n^2 \log(n) \log(B))$ over all lifting steps. The rational number reconstruction, using the Extended Euclidean Algorithm componentwise, has a cost of $O(n \log^2(B))$ operations. We also have $\log(B) = \log(2\|A\|_2^{2n-1}\|b\|_2) = O(n \log(n))$, so the bit complexity is

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

In practice a word sized prime that does not divide $det(A)$ can often be identified by randomly selecting a prime. In the case that a prime is selected that does divide $det(A)$ this can be recognized in the first step of the algorithm when computing $A^{-1} \bmod p$. In general the size of p will depend on the dimension and size of entries in A. In [30] an algorithm to determine a prime p with size $O(\log n + \log\log \|A\|_{\max})$ is analyzed in Corollary 36.

More general complexity analysis of Dixon's method is given by Mulders and Storjohann as Theorem 20 in [22], which does not assume constant bounds on A, b and p. In a later paper [23] the same authors also analyze Dixon's algorithm using fast arithmetic as Proposition 31.

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. 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 a geometric frequency, namely at steps i where $i = 2^k$ 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. We will use $\log(S) = size(A^{-1}b)$ to represent the size of the solution. Recall that we have defined $size()$ of a vector to be the maximum bitsize over all of its entries after all entries are represented with a common denominator. We also know that the numerator and denominator bounds from Cramer's rule and the Hadamard bound gave us $2S \le B = 2\|A\|_2^{2n-1}\|b\|_2$. We will now give an analysis of the complexity of Algorithm 3 in terms of the system dimension n and the solution size $\log(S)$. For simplicity of presentation we assume entries of A, b, p are bounded by a constant.

Algorithm 3 Output Sensitive Dixon Algorithm

```
Input: $A, b, p$        {$Ax = b$ is system to be solved, $p$ is a prime not dividing $det(A)$}
Compute $A^{-1} \bmod p$
$\hat{x} := 0, i := 0, d := b$
while solution not found do
    $y := A^{-1}d \bmod p$
    $\hat{x} := \hat{x} + yp^i$
    if reconstruction step then
        $x :=$Reconstruct($\hat{x}, p^{i+1}$)        {Using Theorem 2.2 component-wise with $B_n = B_d = \sqrt{p^{i+1}/2}$}
        Check $Ax = b$
    end if
    $d := \left(\frac{d-Ay}{p}\right)$
    $i := i + 1$
end while
Return: $x$                                {Solution to system}
```

Theorem 3.2. *Let $Ax = b$ be an n dimensional square nonsingular integer system of equations and suppose the entries of A, b are bounded by a constant. Also suppose that p is a prime bounded by a constant which does not divide $\det(A)$ and $\log(S) = size(A^{-1}b)$. Then the Output Sensitive Dixon Algorithm terminates after $O(n^3 + n^2\log(n)\log(S))$ bit operations.*

Proof. Reducing $A \bmod p$ and computing $A^{-1} \bmod p$ will require $O(n^3)$ operations.

Next we will show that the number of loops is $O(\log(S))$. In the i^{th} loop of the algorithm a solution to the system modulo p^{i+1} will be constructed. By Theorem 2.2 the reconstruction routine is guaranteed to succeed when both B_n and B_d exceed the (unknown) quantity S. Therefore if reconstruction is attempted at a loop where $B_n = B_d = \sqrt{p^{i+1}/2} \ge S$, or $i \ge 2(\log(S)/\log(p))$, the correct solution is ensured to be correctly reconstructed. The geometric choice of reconstruction frequency ensures we will perform at most two times the necessary number of loops beyond the earliest loop where i is large enough to correctly reconstruct the solution.

The number of operations performed in each loop, excluding the cost of the rational reconstruction attempts and the solution check, is $O(n^2\log(n))$ as in the standard Dixon Algorithm. The computational cost of performing rational reconstruction while in loop i is $O(ni^2)$ because each component will have bitsize $O(i)$. In order to check the reconstructed candidate solution x we will first transform to a representation having a common denominator d to get $x = z/d$. Then if z or d exceed the numerator and denominator bounds B_n, B_d the check is aborted, otherwise the solution is checked by computing Az and bd. The cost of computing Az and bd will be $O(n^2\log(n)i)$ since it requires performing an integer 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 combined cost of rational reconstruction and solution checking:

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

Using $\log(S) = O(n\log(n))$ we have the following bound on the total number of bit operations $O(n^3 + n^2\log(n)\log(S) + n\log^2(S) + n^2\log(n)\log(S)) = O(n^3 + n^2\log(n)\log(S))$. □

We see that this algorithm gives an improved run time if the solution size is small is small. Moreover, under the assumption that sizes of A, b, p are bounded by a constant, $\log(S) = O(n\log n)$ so this matches the worst case bound of $O(n^3\log^2 n)$ given in Theorem 3.1.

4 Output Sensitive Iterative Refinement

The iterative-refinement method solves linear systems of equations over the rational numbers by calculating a highly accurate approximate solution and applying Theorem 2.1 to reconstruct the rational solution. The approximate solution is calculated and iteratively refined using numerical methods.

This general idea was used by [31], and was later improved upon by Wan [33] who showed how to more efficiently keep track of the error by rounding and adjusting the approximate solution. 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 size of

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 $\bar{x} \approx \alpha\hat{x}$    {Round to the nearest integer}
    $\Delta := \alpha\Delta - A\bar{x}$    {Update the residual}
    $D := D \times \alpha$    {Update the denominator}
    $N := N \times \alpha + \bar{x}$
end while
Reconstruct $x$ using $N/D$
Return: $x$    {Solution to system}
```

the rational solutions. Algorithm 4 gives a description of the iterative-refinement method similar to the algorithm of Wan [33]. All versions of the iterative-refinement method described in this section require the assumption that the matrix can be successfully numerically factored or inverted. 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. This strategy is described as Algorithm 5.

Algorithm 5 Output Sensitive I. R. 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}
$\Delta := b$    {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 $\bar{x} \approx \alpha\hat{x}$    {Round to the nearest integer}
    $\Delta := \alpha\Delta - A\bar{x}$    {Update the residual}
    $D := D \times \alpha$    {Update the denominator}
    $N := N \times \alpha + \bar{x}$
    if reconstruction step then
        $x :=$Reconstruct$(N, D)$    {Using Theorem 2.1 and $B_d = \sqrt{D/2}$}
        Check $Ax = b$
    end if
end while
Return: $x$    {Solution to system}
```

If rational reconstruction is attempted component-wise at an intermediate step of the iterative-refinement algorithm then by Theorem 2.10 some steps of the EEA will be correct, even if the reconstructed solution is not correct. Therefore the strategy of Algorithm 5 may recompute the same leading sequence of convergents each time rational reconstruction is attempted. Algorithm 6 describes a procedure to warm start rational reconstruction within the output sensitive iterative-refinement method in order to avoid this recomputation. It differs from the previous algorithms discussed because it interweaves the rational reconstruction routine with the refinement steps instead of calling rational reconstruction as a separable routine.

Algorithm 6 O.S. I.R. Method with Warm Starts

```
Input: $A, b$    {$Ax = b$ is system to be solved}
Compute numerical LU factorization of A
$D := 1$    {Common denominator of approximation}
$(Q_0^i, r_0^i, r_1^i) = (I_2, 0, 1) \quad \forall i \in 1, \ldots, \dim(A)$    {Here $Q_k^i, r_k^i, r_{k+1}^i$ represents the elements of the matrix and remainder sequence of the EEA of the $i^{th}$ solution component after $k$ iterations of the EEA.}
$\Delta := b$    {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 $\bar{x} \approx \alpha\hat{x}$    {Round to the nearest integer}
    $\Delta := \alpha\Delta - A\bar{x}$    {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 $\bar{x}_i, \alpha$ and Lemma 4.1
    Perform additional steps of EEA on $(Q_k^i, r_k^i, r_{k+1}^i)$ maintaining $q_{k-1}^i < \sqrt{D/2}$
    {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}
```

After performing each step of the while loop we obtain a refinement of the approximation of the solution to the system of equations. We will use N_i/D to represent the approximation of the i^{th} component, and note that this notation does not appear in the algorithm description because N/D is stored in terms of its EEA matrix and remainder sequence instead of explicitly. For the discussion here we assume that the numerical solver is providing enough correct solution bits that at any step of the algorithm $|N_i/D - (A^{-1}b)_i| < 1/D$ holds.

For the i^{th} component of the solution Q_k^i, r_k^i, r_{k+1}^i stores the matrix and remainder sequence representation of N_i/D after k steps of the EEA. Recall that in Remark 2.3 we saw that the matrix sequence stores the continued fraction convergents of a number. These values are initialized as $(Q_0^i, r_0^i, r_1^i) = (I_2, 0, 1)$. By Theorem 2.10 if $|N_i/D - (A^{-1}b)_i| < 1/D$ and if we update the matrix sequence maintaining $q_{k-1}^i < \sqrt{D/2}$ then either Q_k^i or Q_{k-1}^i will be a correct element of the EEA matrix sequence for the true value of the i^{th} solution component $(A^{-1}b)_i$. Thus, if $k \geq 1$ we can be safe and backtrack to Q_{k-1}^i which will be in the matrix sequence when the EEA is applied to the numerator and denominator of $(A^{-1}b)_i$.

After performing each refinement step two steps must be done to update the continued fraction approximation of the final solution. First Q_k^i, r_k^i, r_{k+1}^i must be updated to reflect the updated representation of the approximation N_i/D. Assuming Q_k^i is found in the matrix sequence for N_i/D we only need to update the remainders r_k^i, r_{k+1}^i. A formula to make this update is given in Lemma 4.1. Secondly, once r_k^i, r_{k+1}^i are updated, additional iterations of the EEA are performed to refine the matrix sequence. This is done component-wise so the progress of the EEA on each component will be different. Steps of the EEA will be performed, starting with Q_k^i, r_k^i, r_{k+1}^i, maintaining $q_{k-1}^i < \sqrt{D/2}$. After performing these operations $x_i = p_{k-1}^i/q_{k-1}^i$ will give the continued fraction approximation of the approximate solution N_i/D with denominator not exceeding $\sqrt{D/2}$.

The following lemma gives an explicit formula for updating the remainders r_k^i, r_{k+1}^i when the approximate solution is refined.

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

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

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

$$\begin{pmatrix} r_k \\ r_{k+1} \end{pmatrix} = \alpha \begin{pmatrix} \hat{r}_k \\ \hat{r}_{k+1} \end{pmatrix} + (-1)^k \bar{x}_i \begin{pmatrix} \hat{q}_{k-2} \\ -\hat{q}_{k-1} \end{pmatrix}.$$

Proof. By Remark 2.3 we have the following

$$\begin{pmatrix} r_k \\ r_{k+1} \end{pmatrix} = Q_k^{-1} \begin{pmatrix} N_i \\ D \end{pmatrix} = \alpha Q_k^{-1} \begin{pmatrix} \hat{N}_i \\ \hat{D} \end{pmatrix} + \bar{x}_i Q_k^{-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$= \alpha \begin{pmatrix} \hat{r}_k \\ \hat{r}_{k+1} \end{pmatrix} + (-1)^k \bar{x}_i \begin{pmatrix} \hat{q}_{k-2} \\ -\hat{q}_{k-1} \end{pmatrix}$$

which gives our proposed formula. □

This gives a way to update the remainder sequence, r_i, r_{i+1}, from $\hat{r}_i, \hat{r}_{i+1}$ without requiring to access N_i or D. The only required information is a scale factor α and difference $\bar{x}_i$, which might have a much smaller representation than N_i or D. After applying Lemma 4.1 to update the remainders more steps of the EEA can be performed advancing the matrix sequence and further refining the continued fraction approximation of each component N_i/D

As noted earlier, the numerator of the approximate solution, which was stored as N in Algorithms 4 and 5, is no longer explicitly stored by Algorithm 6. Instead, it is represented by the matrix sequence and remainders which are updated at each step. From Remark 2.3 we see the approximation of the i^{th} solution component, represented by N_i/D, is stored as

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

In order to check the correctness of the candidate solutions computed in Algorithm 6 the quick check given by Lemma 2.6 can be used at every step, this check is fast to compute but may fail to recognize a correct solution. A more expensive but always correct exact check can be done at loops that are done in a geometric progression ensuring the algorithm terminates after $O(log(S))$ loops. Within this framework there are other choices that could be made regarding how to check the solutions. A suggestion of one referee was to entirely skip the quick checks by Lemma 2.6 and only update Q_k^i, r_k^i, r_{k+1}^i at loops that are a power of two, or some other geometric frequency, to take advantage of asymptotically faster steps of the EEA and performing the full precision checks at these steps.

Remark 4.2. *If A is an $n \times n$ matrix which can successfully be numerically factored, the entries of A, b are bounded by a constant and $\log(S) = size(A^{-1}b)$ then the output sensitive versions of the iterative-refinement methods described as Algorithm 5 and 6 will both 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 in Theorem 3.2, so we only note some differences here. In [33] Wan gave a proof of correctness and an analysis of his algorithm which is similar to Algorithm 4. We have stated this result as an informal remark and refer the reader to Wan's paper [33] to see how one can make a more rigorous statement of this type involving a numerical solver. The only significant difference between Algorithm 4 and its output sensitive counterparts is how and when the rational reconstruction is performed. Moreover, by using theorem 2.10 to warm start the rational reconstruction at each step Algorithm 6 will perform asymptotically the same amount of computation for rational reconstruction as Algorithm 5. In Algorithm 6 the reconstructed solution is available at every step of the while loop and thus the quick check to certify $Ax = b$ given by Lemma 2.6 can be done at every loop of the refinement procedure. The more computationally expensive check will only be performed at a geometric frequency.

We note that for p-adic lifting, the idea of using warm starts for the EEA 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. Testing the iterative-refinement solver allows us to evaluate the warm starts within the iterative-refinement method as described in Algorithm 6 in comparison with the standard and output sensitive methods (Algorithms 4 and 5). Moreover, in [8] 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.

Our implementation of Dixon's algorithm is written in C/C++ and uses the FFLAS and FFPACK packages [12], 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. For Dixon's Algorithm any non-integral inputs are scaled to be integral before solving.

The implementation of the iterative-refinement methods are written in C and uses the BLAS [10, 18] and LAPACK [2] routines for the dense numerical linear algebra. We have used the ATLAS package [35] 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 using the framework of Algorithm 5; third, we implement a version of Algorithm 6 where the partially reconstructed solutions are stored from step to step and reused.

We use a straightforward implementation of rational reconstruction, employing a technique referred to as the DLCM method in Section 3.2.3 of [8]. This technique is also used by Chen and Storjohann [6, 7] and others [16, 29]. The DLCM method 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 common denominator grows too large. For Dixon's method we apply the DLCM method as it is described in [6, 7, 8]. For the iterative-refinement methods we use a modified strategy because the warm starts in Algorithm 6 store the work of the EEA from step to step making it incompatible with the DLCM method. The modified strategy we use is to reconstruct the candidate exact solution component-wise and keep track of the common denominator of the re-constructed components during each reconstruction step. If this common denominator of the reconstructed components exceeds the denominator bound ($\sqrt{D/2}$ where D is the denominator of the approximate solution) then no further steps of the EEA are performed and more refinement steps are performed. This gives a minor slowdown to Algorithms 4 and 5 but allows a side by side comparison between them and Algorithm 6. An recent study of vector rational number reconstruction can be found in [3].

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 [11] or IML [6, 7].

5.2 Test Problems

The goal of our computational experiments is two fold. First, 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. Secondly, 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	$\{L_n\}_{ij} = \min(i, j)/\max(i, j)$

There was some difficulty in choosing which problems to consider. It is difficult to find an explicit linear system of equations for which the size of the solution meets the Hadamard bound exactly. The Hadamard matrices have determinants which meet the Hadamard determinant bound tightly $det(D_n) = 2^{n-1}$. However, when using these matrices the solution size will not be as large because the inverse matrix $D_n^{-1} = \frac{1}{n} D_n$ has small entries.

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 frequently 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 size 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. In addition to the total solution time for each method we include a profile of how time was spent in different stages of the algorithm. The solution time is divided between the following three tasks: the finite field

Table 2: Solve times for Dixon algorithms in seconds

Problem Details			Standard Dixon (Alg. 2)				Output Sensitive Dixon (Alg. 3)			
Matrix	log(B)	log(S)	**Total**	Factor	Lift	R.R.	**Total**	Factor	Lifting	R.R.& S.V.
D_{1024}	12284	10	**28.34**	0.42	27.91	0.01	**1.12**	0.43	0.07	0.62
D_{2048}	24572	11	**242.65**	2.74	239.89	0.01	**5.51**	2.71	0.29	2.51
D_{4096}	57339	12	**2273.67**	20.01	2253.61	0.05	**32.68**	21.20	1.28	10.19
R_{500}	13988	13271	**9.98**	0.19	9.63	0.17	**9.96**	0.17	9.60	0.18
R_{1000}	27988	26557	**75.51**	0.86	73.68	0.97	**75.26**	0.90	73.33	1.02
R_{2000}	55988	53137	**582.20**	4.34	571.98	5.88	**582.87**	4.43	572.35	6.09
H_{500}	1432292	1269	**5916.43**	0.17	5916.02	0.24	**6.40**	0.18	5.89	0.33
H_{1000}	5742567	2540	**204945.46**	0.84	204941.63	2.99	**84.96**	0.84	82.15	1.98
H_{2000}	22997129	5084	**-**	-	-	-	**1218.51**	4.37	1196.43	17.71
V_{100}	130944	1046	**13.72**	0.01	13.68	0.03	**0.17**	0.01	0.07	0.09
V_{300}	1474141	4079	**4049.52**	0.06	4046.52	2.93	**8.74**	0.06	4.62	4.05
V_{500}	4469528	7530	**67972.23**	0.18	67947.28	24.77	**61.02**	0.18	36.56	24.28
L_{500}	727997	3	**218.41**	0.19	218.20	0.02	**0.33**	0.19	0.01	0.13
L_{1000}	2885996	3	**3043.20**	0.88	3042.15	0.17	**1.48**	0.91	0.03	0.54
L_{2000}	11531996	3	**52873.03**	4.36	52867.53	1.14	**6.87**	4.58	0.14	2.15

matrix factorization, the p-adic lifting steps, and the rational reconstruction (including solution verification). Solution verification is only performed when rational reconstruction is attempted by the output sensitive methods at loops where the correctness is not guaranteed. Whenever p^i surpasses the bound B solution checks are not necessary. Due to system load and other factors solution times vary with each run, therefore these timings should be considered as approximate values, the reason times are shown to 1/100 of a second is to allow for a comparison between the subroutines. The table also includes the log of the Hadamard bound on the solution size $\log(B) = \log(2\|A\|_2^{2n-1}\|b\|_2)$, along with the actual size $\log(S) = size(A^{-1}b) \leq \log(B) - 1$.

The first observation we make from Table 2 is that the Hadamard bound was a very weak upper bound on the solution size on all problem classes except the randomly generated matrices. On the set of randomly generated matrices, the Hadamard bound did provide a fairly tight bound on the final solution bitsize. In these cases the output sensitive algorithm performs the same number of loops as the standard Dixon algorithm and also performs additional reconstruction attempts at intermediate steps. Even on these problems, the output sensitive Dixon algorithm has approximately the same solution times as the standard Dixon algorithm. This demonstrates that even if the Hadamard bound is nearly tight the output sensitive lifting only performs a small amount of additional computation. The occurs because at steps where incorrect solutions are generated, this incorrectness can be recognized very quickly. The first possibility is that the rational reconstruction routine aborts early if the common denominator of the reconstructed components grows too large. The second possibility is that the algorithm reconstructs an incorrect solution vector and checks its correctness by evaluating the system of equations. In this second case it will likely only evaluate a very small number of equations to recognize its incorrectness, which could be much less costly than evaluating the full system as was accounted for in the worst case complexity analysis of Theorem 3.2. For the remainder of the problem set the output sensitive method has an advantage of several orders of magnitude. The lifting steps that were avoided gave a significant reduction in the computational costs. We also notice on some problems with smaller solution size the cost of rational reconstruction and solution verification is larger for the output sensitive method than the standard methods due to the final solution verification.

Results for the iterative-refinement based solvers using Algorithms 4, 5 and 6 on the Hadamard and random matrices are given in Table 3. The Hilbert, Vandermonde and Lehmer matrices are were too numerically difficult for the LAPACK routines to handle so they are not included in the results. We observe from Table 3 that the performance ratio

Table 3: Solve times for iterative-refinement in seconds

	Lifting Strategy (Algorithm Number)		
Matrix	Std. (4)	O.S. (5)	O.S. & W.S. (6)
D_{1024}	8.57	0.91	0.92
D_{2048}	71.10	4.49	4.57
D_{4096}	965.87	27.39	27.17
R_{500}	12.12	12.07	12.46
R_{1000}	94.52	94.60	98.25
R_{2000}	763.50	756.89	798.66

between the standard and output sensitive lifting strategies of Algorithms 4 and 5 is similar to their related versions of the Dixon method compared in Table 2. We also observe that Algorithm 6 did not give any improvement over the basic output sensitive lifting strategy given in Algorithm 5. The extra bookkeeping required in Algorithm 6 may be more costly than any benefits gained by saving the information for these problems. In comparison to the breakdown for Dixon's method in Table 2 for the factorization, refinement and reconstruction there was extra cost was incurred by the reconstruction steps in these algorithms due to the less efficient handling of the DLCM method that was done to allow a side by side comparison between Algorithms 4, 5 and 6.

The main purpose of these computations was to study the effectiveness of output sensitive methods, and not to compare Dixon's algorithm vs. the iterative-refinement method. However, we can make some observations about their relative speed. Before making the direct comparison we note one difference in the implementation: as described in Section 5.1 the implementations handle the DLCM method differently because the version of this algorithm used in Dixon's method is not compatible with the warm starts in Algorithm 6. If Algorithm 5 is adjusted to use the same version of the DLCM method as Algorithm 3 we observed that the solution times of Algorithm 5 were often faster on our test set, but not by orders of magnitude. Therefore we conclude that the iterative-refinement method can be faster than Dixon's Algorithm when both are applicable, but not by a huge margin. We also remark that tuning parameters such as how large of a prime p is used for p-adic lifting or how many digits of accuracy the numerical solver has in iterative refinement can effect the solution times. Other recent studies [8, 33] observed the iterative-refinement method to be slightly faster but not by a huge order of magnitude.

6 Conclusion

Our study reinforces a conclusion that has already been observed in practice: output sensitive lifting can improve algorithms for symbolically solving systems of linear equations. We show that output sensitive algorithms can allow for systems of rational linear equations to be solved very quickly when the final solutions are small in size, while maintaining the same worst case bit complexity even when solutions are large in size. Tests were performed on several types of dense systems where output sensitive lifting was observed to give significant improvements on problems with small solution size, without noticeable slowdown even when the solution size was large.

We introduced a strategy to warm start the rational reconstruction portion of the iterative-refinement method. While this did not further improve on the other output sensitive version of iterative refinement there may be other settings in which warm starting the EEA or rational reconstruction could prove helpful.

We have primarily focused on output sensitive lifting applied to dense systems of equations; this technique is also fully applicable in the sparse setting. Our results suggest that any exact precision linear system solver relying on iterative methods should employ output sensitive lifting.

Acknowledgments

The author would like to thank William Cook, Helder Inacio and the anonymous referees for helpful suggestions that improved the quality of this paper.

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] C. Bright. Vector rational number reconstruction. Master's thesis, School of Mathematics, University of Waterloo, 2010.

[4] 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.

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

[6] 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.

[7] 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.

[8] W. J. Cook and D. E. Steffy. Solving very sparse rational systems of equations. *ACM Transactions on Mathematical Software*, To Appear.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] E. Krishnamurthy, T. Rao, and K. Subramanian. p-adic arithmetic procedures for exact matrix computations. *Proc. of the Indian Academy of Sciences, Series A*, 82:165–175, 1975.

[18] 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.

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

[20] 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.

[21] R. Moenck and J. Carter. Approximate algorithms to derive exact solutions to systems of linear equations. 72:65–73, 1979.

[22] 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.

[23] T. Mulders and A. Storjohann. Certified dense linear system solving. *Journal of Symbolic Computation*, 37:485–510, 2004.

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

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

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

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

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

[29] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl/`, 2008.

[30] A. Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21:609–650, 2005.

[31] 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.

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

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

[34] P. S. Wang. A p-adic algorithm for univariate partial fractions. In *SYMSAC '81: Proceedings of the fourth ACM symposium on Symbolic and algebraic computation*, pages 212–217, New York, NY, USA, 1981. ACM.

[35] 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.

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

Appendix

More general complexity analysis of Dixon's method is given by Mulders and Storjohann as Theorem 20 in [22], which does not assume constant bounds on A, b and p.

Theorem 6.1. *The p-adic lifting algorithm for solving a system of integer equations exactly is correct and given input A, b, p it will terminate after*

$$O(n^3(\log n + \log \|A\|_{\max} + \log p)^2 + n \log^2 \|b\|_\infty)$$

bit operations.

Their statement of the algorithm differs slightly from Algorithm 2 but follows the same basic structure. Mulders and Storjohann prove this theorem using standard arithmetic and in a later paper [23] the they also give a detailed complexity analysis of Dixon's method using fast arithmetic as Proposition 31.

We will now analyze Algorithm 3 without assuming constant size bounds on entries of A, b and p. This is similar to the proof of Theorem 20 in [22].

Theorem 6.2. *Let $Ax = b$ be a square nonsingular integer system of equations and a prime p is known not dividing $det(A)$ and $\log(S) = size(A^{-1}b)$ then the Output Sensitive Dixon Algorithm terminates after*

$$O\left(n^3 \log^2(p) + n^2 \log(\|A\|_{\max}) \log(p) + n \log(S) \log(\|b\|_\infty) \right.$$
$$\left. + n^2 \log(S)(\log n + \log \|A\|_{\max} + \log p)\right)$$

bit operations.

Proof. We will bound the complexity of this algorithm in three steps. First we look at the initial cost of reducing A mod p and computing its mod p inverse. Second we consider the cost of all of the lifting steps. Third, we consider the combined cost of performing rational reconstruction including checking the intermediate solutions.

Inversion: Reduction of $A \mod p$ and computation of $A^{-1} \mod p$ can be done using

$$O(n^2 \log(\|A\|_{\max}) \log(p) + n^3 \log^2(p))$$

bit operations.

Lifting: The number of loops the algorithm will perform is $O(\log(S)/\log(p))$. We will now count the cost of each lifting loop.

At the i^{th} step of the algorithm $d = (b - A(A^{-1}b \bmod p^i))/p^i$ and since p^i must divide $b - A(A^{-1}b \bmod p^i)$ this implies d always has entries with absolute value at most $\|b\|_\infty + n\|A\|_{\max}$. To compute $y := A^{-1}d \bmod p$ we reduce d mod p which will cost $O(n \log(p)(\log \|b\|_\infty + \log n + \log \|A\|_{\max}))$ and then doing a mod p matrix-vector multiplication will cost $O(n^2 \log^2(p))$.

Now we bound the cost of computing $\hat{x} := \hat{x} + yp^i$ in each loop. Observe that $\hat{x}$ will require at most $O(\log S)$ bits to represent it at any stage of the algorithm. p^i can be updated and stored from step to step and will always have size $O(\log(S))$. y will have size $O(\log p)$. The dominating cost will be the multiplication of yp^i which will cost $O(n \log(S) \log(p))$.

Finally we consider the cost of updating $d := (d - Ay)/p$. Since y has entries with absolute value at most p, and entries of Ay are at most $n\|A\|_{\max}p$, the cost of the multiplications and additions required to compute Ay is bounded by $O(n^2(\log(\|A\|_{\max}) \log(p) + \log n))$. Subtracting Ay from d will cost $O(n(\log n + \log p + \log \|A\|_{\max} + \log \|b\|_\infty))$ and then dividing by p will have cost $O(n \log(p)(\log \|b\|_\infty + \log n + \log \|A\|_{\max} + \log p))$.

Combining terms we have the following bound on the computation required in each loop:

$$O(n \log(p) \log(S) + n \log(p)(\log n + \log \|b\|_\infty)$$
$$+ n^2(\log(\|A\|_{\max}) \log(p) + \log n + \log^2(p))$$

Multiplying the total loop cost by the number of loops $O(\log(S)/\log(p))$ can be bounded by:

$$O(n \log^2(S) + n \log(S) \log(\|b\|_\infty)$$
$$+ n^2 \log(S)(\log(\|A\|_{\max}) + \log n + \log p))$$

as a bound on all computation in the lifting steps.

Reconstruction: Finally we consider the combined cost of the rational reconstruction and solution checks performed. At the i^{th} loop the cost of rational reconstruction is bounded by $O(n(i \log(p))^2)$ since each of the n components will have bitsize $O(i \log(p))$. Once a candidate solution x is reconstructed it is checked for correctness. This can be done by first transforming it to be represented with a common denominator as $z/d = x$. If any entries of z or the common denominator d exceed the numerator and denominator bounds $B_n = B_d = \sqrt{p^{i+1}/2}$ then the check is aborted. Now, if the bitsize of d and the entries of z are $O(i \log(p))$ then the cost of computing Az will be $O(n^2(\log(\|A\|_{\max})(i) \log(p) + \log(n)))$ since it requires performing an integer matrix-vector multiplication where the entries of the matrix are bounded by $\|A\|_{\max}$, and the entries of the vector have bitsize $O(i \log p)$ and the largest entries of the resulting values of Az have bitsize bounded by $O(\log \|A\|_{\max} + \log n + (i) \log(p))$. The cost of computing bd is $O(n \log(\|b\|_\infty)(i) \log(p))$. Therefore the total cost of rational reconstruction and solution checking over all loops $k = 1, 2, \ldots, O(\log(\log(S)/\log(p)))$ where it is applied is:

$$\sum_{k=1}^{O(\log(\log(S)/\log(p)))} O\left(n \log^2(p)(2^k)^2\right.$$
$$+ n^2(\log(\|A\|_{\max}) \log(p) 2^k + \log n)$$
$$\left. + \, n \log(\|b\|_\infty) \log(p) 2^k\right).$$

This summation is bounded above by:

$$O(n \log^2(S) + n^2 \log(S)(\log \|A\|_{\max} + \log n)$$
$$+ n \log(S) \log(\|b\|_\infty))$$

Total cost: Finally, considering the cost of the matrix inversion, lifting and reconstruction attempts can all be bounded by:

$$O\left(n^3 \log^2(p) + n^2 \log(\|A\|_{\max}) \log(p) + n \log^2(S)\right.$$
$$+ n^2 \log(S)(\log n + \log \|A\|_{\max} + \log p)$$
$$\left. + n \log(S) \log(\|b\|_\infty)\right)$$

And using the fact that $\log(S) = O(n \log(n) + n \log(\|A\|_{\max}) + \log \|b\|_\infty)$ we can simplify this to:

$$O\left(n^3 \log^2(p) + n^2 \log(\|A\|_{\max}) \log(p) + n \log(S) \log(\|b\|_\infty)\right.$$
$$\left. + n^2 \log(S)(\log n + \log \|A\|_{\max} + \log p)\right)$$

which gives the desired bound. □

This result agrees with the result of Theorem 3.1 when the sizes of the input numbers are all treated as constants. Comparing this to Theorem 6.1 we see that the algorithm can perform asymptotically faster if the final solution size is small. We will also see that it performs no worse even when the solution size is as large as possible.

Corollary 6.3. *The bit complexity of the Output Sensitive Dixon Algorithm as given by Theorem 6.2 is*

$$O(n^3(\log n + \log \|A\|_{\max} + \log p)^2 + n \log^2 \|b\|_\infty)$$

Proof. The result of Theorem 6.2 gives the following bound:

$$O\left(n^3 \log^2(p) + n^2 \log(\|A\|_{\max}) \log(p) + n \log(S) \log(\|b\|_\infty)\right.$$
$$\left. + n^2 \log(S)(\log n + \log \|A\|_{\max} + \log p)\right)$$

By the Hadamard bound we know that $\log(S) = O(n\log(n) + n\log(\|A\|_{\max}) + \log \|b\|_\infty)$, so plugging in this bound and removing all terms that already satisfy the desired bound we are left with:

$$O(n^2 \log(\|b\|_\infty)(\log n + \log \|A\|_{\max} + \log p))$$

To observe that these terms also meet our desired bound consider the two cases where either $\log(\|b\|_\infty) \geq n(\log n + \log \|A\|_{\max} + \log p)$ or $\log(\|b\|_\infty) \leq n(\log n + \log \|A\|_{\max} + \log p)$ and the result follows. □