Approximate Integer Common Divisors

Nick Howgrave-Graham

IBM T.J.Watson Research Center
PO Box 704, Yorktown Heights, New York 10598, USA
nahg@watson.ibm.com

Abstract. We show that recent results of Coppersmith, Boneh, Durfee and Howgrave-Graham actually apply in the more general setting of (partially) approximate common divisors. This leads us to consider the question of "fully" approximate common divisors, i.e. where both integers are only known by approximations. We explain the lattice techniques in both the partial and general cases. As an application of the partial approximate common divisor algorithm we show that a cryptosystem proposed by Okamoto actually leaks the private information directly from the public information in polynomial time. In contrast to the partial setting, our technique with respect to the general setting can only be considered heuristic, since we encounter the same "proof of algebraic independence" problem as a subset of the above authors have in previous papers. This problem is generally considered a (hard) problem in lattice *theory*, since in our case, as in previous cases, the method still works extremely reliably in practice; indeed no counter examples have been obtained. The results in both the partial and general settings are far stronger than might be supposed from a continued-fraction standpoint (the way in which the problems were attacked in the past), and the determinant calculations admit a reasonably neat analysis.
Keywords: Greatest common divisor, approximations, Coppersmith's method, continued fractions, lattice attacks.

1 Introduction

When given new mathematical techniques, as is the case in [1] and [5], it is important to know the full extent to which the result can be used, mathematically, even if this generalisation does not have immediate applications (to cryptography, say). In this paper we will start by describing approximate common divisor problems. Later we will show that the above results can be seen as special instances of this problem, and we will describe a lattice based solution to (a version of) the general problem. For now let us just concentrate on explaining approximate common divisor problems.

As an example, we explain this in the more specific and familiar case of *greatest* common divisors. If we are given two integers a and b we can clearly find their gcd, d say, in polynomial time. If d is in some sense large then it may be possible to incur some additive "error" on either of the inputs a and b, or both, and still recover this gcd. This is what we refer to as an approximate common divisor problem (ACDP); although we delay its rigorous definition to later. Of course if there is too much error incurred on the inputs, the algorithm may well not be able to discern the gcd d we had initially over some other approximate divisors d' (e.g. they may all leave residues of similar magnitude when dividing a and b). In this sense, the problem is similar to those found in error correcting codes.

Continuing this error correcting code analogy we can state the problem from the standpoint of the design of the decoding algorithm, i.e. we wish to create an algorithm which is given two inputs a_0 and b_0, and bounds X, Y and M for which one is assured that $d|(a_0 + x_0)$ and $d|(b_0 + y_0)$ for some $d > M$ and x_0, y_0 satisfying $|x_0| \leq X$, $|y_0| \leq Y$. The output of the algorithm should be the common divisor d, or all of the possible ones if more than one exists. We explore the following questions: under what conditions on these variables and bounds

– is d uniquely defined, or more generally limited to polynomially many solutions?
– does an algorithm exist to recover these d in polynomial time?

Without loss of generality let us assume that our inputs are ordered so that $X \geq Y$. If one of the inputs is known exactly, i.e. $Y = 0$, then we call this a *partially approximate common divisor problem* (PACDP), and we refer to an algorithm for its solution as a PACDP algorithm. If however neither input is known exactly, i.e. $Y > 0$, then we refer to the problem as a *general approximate common divisor problem* (GACDP), and an algorithm for its solution is called a GACDP algorithm.

In section 3 we show that the results given in [1] and [5] may be seen as special instances of a PACDP, and the techniques used therein effectively give a PACDP algorithm.

As a motivating example of this, consider the widely appreciated result in [1], which states that if $N = pq$ where $p \sim q \sim \sqrt{N}$, and we are given the top half of the bits of p then one can recover the bottom half of the bits of p in polynomial time. Notice that in effect we are given p_0, such that $p_0 = p + x_0$, for some x_0 satisfying $|x_0| < N^{1/4}$, from which we can recover the whole of p. It is not so widely known that the result also holds if we are given any integer p'_0 such that $p'_0 = kp + x_0$ for some integer k (and the same bound on x_0 as before), i.e. we can still recover all the bits of p from this information too.

This immediately shows that Okamoto's cryptosystem [11] leaks the private information (the factorisation of $n = p^2q$) from the public information (n and $u = a + bpq$ where $a < (1/2)\sqrt{pq}$) in polynomial time[1], so this result can be considered an even stonger break than that given in [14], which recovers all plaintexts, but does not recover the secret information.

In section 4 we go on to consider the GACDP, and produce some new and interesting bounds for polynomial time algorithms which (heuristically) solve this. For ease of analysis we do restrict our attention to the case when $a_0 \sim b_0$

[1] In fact the size of a is much smaller than it need be for this attack to work.

and $X \sim Y$ (we call such input "equi-sized"), though similar techniques can brought to bear in more general situations.

This general problem is thought to be very interesting to study from a mathematical point of view, and the lattice analysis is also considered to be quite neat and interesting from a theoretical standpoint, however this generalisation admittedly lacks an obvious (useful) application in either cryptography or coding theory. It is hoped that with this presentation of the problem, a use will subsequently be found.

In section 7 we conclude with some open and fundamental questions associated with these general lattice techniques.

1.1 Presentation and Algorithm Definitions

In the remainder of this paper a and b will always denote integers which do have a "large" common divisor, and d will be used to represent this common divisor. The (known) approximations to a and b will be denoted by a_0 and b_0 respectively, and their differences by $x_0 = a - a_0$, $y_0 = y - y_0$. A real number $\alpha \in (0 \ldots 1)$ is used to indicate the quantity $\log_b d$, and α_0 is used to indicate a (known) lower bound for this quantity.

Given two integers u, v we will write $u \sim_{\varepsilon_0} v$ if $|\log_2 \log_2 u - \log_2 \log_2 v| < \varepsilon_0$, though we frequently drop the ε_0 subscript, when it is clear what we mean.

We now define the algorithms to solve the ACDPs presented in this paper. We include their bound information to fully specify the algorithms, though the proof of these bounds, and the proof that their output is polynomially bounded, is left to their respective sections. We start with the PACDP algorithms described in sections 2 and 3.

Algorithm 11. *The (continued fraction based)* partially approximate common divisor algorithm `PACD_CF`*, is defined thus: Its input is two integers a_0, b_0 such that $a_0 < b_0$. The algorithm should output all integers $d = b_0^{\alpha}$, $\alpha > 1/2$, such that there exists an x_0 with $|x_0| < X = b_0^{2\alpha-1}$, and d divides both $a_0 + x_0$ and b_0, or report that no such d exists (under the condition on X we are assured that there are only polynomially many solutions for d).*

Algorithm 12. *The (lattice based)* partially approximate common divisor algorithm `PACD_L`*, is defined thus: Its input is two integers a_0, b_0, $a_0 < b_0$ and two real numbers $\varepsilon, \alpha_0 \in (0 \ldots 1)$. Let us define $M = b_0^{\alpha_0}$ and $X = b_0^{\beta_0}$ where $\beta_0 = \alpha_0^2 - \varepsilon$. The algorithm should output all integers $d > M$ such that there exists an x_0 with $|x_0| < X$, and d divides both $a_0 + x_0$ and b_0, or report that no such d exists (under the conditions on M and X we are assured that there are only polynomially many solutions for d).*

Firstly notice that the condition $a_0 < b_0$ is not a limitation at all. Since we know b_0 exactly we may subtract any multiple of it from a_0 to ensure this.

Secondly (and far more importantly) notice that there is a subtle distinction between algorithms 11 and 12 in that in the continued fraction based algorithm

α is not an input, but rather is defined in terms of any common divisor d (this is true of the GACDP algorithms defined below too). This is preferential to the situation with the lattice algorithms, in which it is presently necessary to state in advance an $\alpha_0 < \alpha$, and then the bound X is defined in terms of α_0 rather than α (so one would wish α_0 to be very close to α to ensure X was as large as possible).

Thirdly notice the requirement for $\alpha > 1/2$ in the continued fraction techniques. A major contribution of this paper is in showing that we may solve the ACDPs for $\alpha < 1/2$ too, using the lattice based methods.

Lastly notice the appearance of ε in the lattice based variants. This is because the bound on X in these algorithms is defined asymptotically. In order to know X explicitly we allow $\varepsilon \ll \log_{b_0} X$ to be given as input to the algorithm.

We give two equi-sized (i.e. $a_0 \sim b_0$) GACDP algorithms in the paper, one in section 2 and one in section 4. The lattice based approach is only defined for $\alpha < 2/3$. Refer to figure 61 to see a graphical representation of the bounds from each of the algorithms.

Algorithm 13. *The* (continued fraction based) equi-sized GACDP algorithm `GACD_CF`*, is defined thus: Its input is two integers a_0, b_0 subject to $a_0 \sim b_0$, $a_0 < b_0$. The algorithm should output all integers $d = b_0^{\alpha}$, $\alpha > 1/2$ such that there exist integers x_0, y_0 with $|x_0|, |y_0| < X = b_0^{\beta}$ where $\beta = \max(2\alpha - 1, 1 - \alpha)$, and d divides both $a_0 + x_0$ and $b_0 + y_0$, or report that no such d exists (under the condition on X we are assured that there are only polynomially many solutions for d).*

Algorithm 14. *The* (lattice based) equi-sized GACDP algorithm `GACD_L`*, is defined thus: Its input is two integers a_0, b_0 subject to $a_0 \sim b_0$, $a_0 < b_0$, and two real numbers $\varepsilon, \alpha_0 \in (0 \ldots 2/3)$. Let us define $M = b_0^{\alpha_0}$ and $X = b_0^{\beta_0}$ where $\beta_0 = 1 - (1/2)\alpha_0 - \sqrt{1 - \alpha_0 - (1/2)\alpha_0^2} - \varepsilon$. The algorithm should output all integers $d > M$ such that there exist integers x_0, y_0 with $|x_0|, |y_0| < X$, and d divides both $a_0 + x_0$ and $b_0 + y_0$, or report that it is unlikely that such a d exists (under the conditions on M and X we are assured that there are only polynomially many solutions for d).*

As the above definitions mention, the number of common divisors d is polynomially bounded when the conditions of the algorithms are met. If one wishes to use the algorithms as encoding/decoding algorithms, and so require a *unique* output from the algorithms, then one should ensure that the "aimed for" solution is substantially below the algorithm bounds, meaning that it is highly (exponentially) unlikely that any of the other (polynomially many) common divisors will be confused with it.

Notice that since algorithm `GACD_L` is heuristic, it is possible that a d exists which meets the conditions of the algorithm, but it is not found. In order to give an indication of the probability of this event happening one can refer to the results in Table 1. As can be seen there, no such occurances were detected, and so such events are considered to be extremely rare.

2 A Continued Fraction Approach

One way to approach solving the ACDPs is to consider the sensitivity of the Euclidean algorithm to additive error of its inputs. This can be studied via the use of coninued fractions, as explained in [6]. One of the many places such an analysis was found useful was in [15], when attacking RSA with a small decrypting exponent, and we look at this further in section 5.

The main results that are useful in this analysis are the following:

Theorem 21. *Let ρ be any real number, and let g_i/h_i, $i = 1 \dots m$ denote the (poynomially many) approximants to ρ during the continued fraction approximation.*

For all $i = 1 \dots m$ we have that

$$\left|\rho - \frac{g_i}{h_i}\right| < \frac{1}{h_i^2}.$$

Moreover for every pair of integers s, t such that

$$\left|\rho - \frac{s}{t}\right| < \frac{1}{2t^2}$$

then the ratio s/t will occur as one of the approximants g_j/h_j for some $j \in (1 \dots m)$.

To see how this applies to creating an ACDP algorithm, let us recall the input to the continued fraction based algorithms, namely two integers a_0 and b_0, $a_0 < b_0$, and we search for common divisors $d = b_0^\alpha$ that divide both $a_0 + x_0$ and $b_0 + y_0$ ($y_0 = 0$ in the PACDP).

In this section we will assume that x_0 and y_0 are such that $|x_0|, |y_0| < b_0^\beta$, and show the dependence of β on α so that the continued fraction approach is assured of finding the common divisors d. We will see that the basic algorithm is essentially the same for `PACD_CF` and `GACD_CF`, i.e. the fact that we know $y_0 = 0$ will only help limit the number of d, not the approach used to find them.

Let a' and b' denote $(a_0 + x_0)/d$ and $(b_0 + y_0)/d$ respectively, so clearly the sizes of a' and b' are bounded by $|a'|, |b'| < b_0^{1-\alpha}$. Also notice that

$$\frac{a_0}{b_0} = \frac{a_0 + x_0}{b_0 + y_0} + \frac{a_0 y_0 - b_0 x_0}{b_0(b_0 + y_0)} = \frac{a'}{b'} + \frac{a_0 y_0 - b_0 x_0}{b_0(b_0 + y_0)}$$

so we have

$$\left|\frac{a_0}{b_0} - \frac{a'}{b'}\right| = \frac{|a_0 y_0 - b_0 x_0|}{b_0(b_0 + y_0)} < b_0^{\beta - 1}$$

This means that by producing the continued fraction approximation of $\rho = a_0/b_0$, we will obtain a'/b' whenever $b_0^{\beta-1} < 1/(2(b')^2)$. Since we are primarily concerned with large a_0 and b_0 we choose to ignore constant terms like 2, and by using the fact that $|b'| < b_0^{1-\alpha}$ we see that this inequality holds whenever $\beta - 1 < 2(\alpha - 1)$, or $\beta < 2\alpha - 1$ as anticipated by the algorithm definitions.

Thus for both PACD_CF and GACD_CF the algorithm essentially comes down to calculating the continued fractions approximation of a_0/b_0; the only difference between these two algorithms is what is then outputted as the common divisors d.

Let us first concentrate on PACD_CF, and let g_i/h_i denote the (polynomially many) approximants in the continued fraction approximation of a_0/b_0. If $d = b_0^\alpha$ divides both $a_0 + x_0$ and b_0 and $|x_0| < X = b_0^{2\alpha-1}$ we know a_0/b_0 is one of these approximants. It remains to test each of them to see if h_i divides b_0; if it does then we output $d = b_0/h_i$ as a common divisor. Note that for all such approximants we are assured that $|x_0| < X$ since $x_0 = a_0 - dg < b_0/h^2 = b_0^{2\alpha-1}$.

This proves the existence of algorithm PACD_CF.

We now turn our attention to algorithm GACD_CF. The first stage is exactly the same, i.e. we again consider the (polynomially many) approximants g_i/h_i, however now h_i need not divide b_0. Instead we find the integer k (which will correspond to our common divisor d) which minimises the max-norm of the vector $k(g_i, h_i) - (a_0, b_0) = (x_0, y_0)$. Again we are assured that for all approximants $|x_0|, |y_0| < X = b_0^{2\alpha-1}$.

However consider $(x_0', y_0') = (k + l)(g_i, h_i) - (a_0, b_0) = (x_0, y_0) + l(g_i, h_i)$ for some integer l. If it is the case that $|lg_i|, |lh_i| < X$ as well, then this will mean $(k + i)$ for all $i = 0 \dots l$ satisfy the properties of the common divisors d, and shoud be outputted by the algorithm. Since $h_i = b_0^{1-\alpha}$ there are $b_0^{3\alpha-2}$ choices for l, which becomes exponential for $\alpha > 2/3$.

In order to limit ourselves to polynomially many d we define $X = b_0^\beta$ where $\beta = \min(2\alpha - 1, 1 - \alpha)$, which ensures there is (exactly) one common divisor d associated with each continued fraction approximant g_i/h_i. This proves the existence of algorithm GACD_CF.

Readers who are reminded of Wiener's attack on RSA by this analysis should consult section 5.

Even though it is known that the continued fraction analysis is optimal for the Euclidean algorithm, we do obtain better bounds for the PACDP and the GACDP in the next sections. This is because we are not restricted to using the Euclidean algorithm to find the common divisors d, but rather we can make use of higher dimensional lattices, as originally done in [1].

3 Using Lattices to Solve PACDP

Essentially there is nothing new to do in this section, except point out that the techniques previously used to solve divisibility problems, e.g. [1] and [5], actually apply to the more general case of PACDPs.

To do this we will concentrate on the approach taken in [5] (because of the simpler analysis), which we sum up briefly below (primarily so that we can draw similarities with the method described in section 4). The reader is encouraged to consult the original paper for complete details of this method.

The algorithm was originally used to factor integers of the form $N = p^r q$. In this section we will restrict our attention to $r = 1$, though we note that the techniques still work for larger r (which would be analagous to ACDPs such that one of the inputs is near a *power* of the common divisor d).

A key observation in [5] is that if we have m polynomials $p_i(x) \in \mathbf{Z}[x]$, and we are assured that for some integer x_0 we have $p_i(x_0) = 0 \bmod t$ for all $i = 1 \dots m$, then any (polynomial) linear combination of these polynomials, i.e. $r(x) = \sum_{i=1}^{m} u_i(x)p_i(x)$ also has the property that $r(x_0) = 0 \bmod t$.

The trick is to find a polynomial $r(x)$ of this form, which is "small" when evaluated at all the "small" integers. Let $r(x) = r_0 + r_1x + \ldots + r_hx^h$. One way to ensure that $|r(x)| \le hX^h$ when evaluated at any x such that $|x| < X$ is to make the r_i satisfy $|r_i|X^i \le X^h$. This is the approach we shall take, and we shall use lattice reduction algorithms to generate these coefficients.

Notice that if t were such that $t > hX^h$ and x_0 were such that $|x_0| < X$, then the integer $r(x_0)$ which must be a multiple of t, but cannot be as large as t, must therefore equal 0. By finding all the roots of the equation $r(x) = 0$ over the integers we therefore find all possible x_0 for which the $p_i(x_0) = 0 \bmod t$ when $|x_0| < (t/h)^{1/h}$.

To see the relevance of this with the PACDP, notice that in the PACDP we are effectively given two polynomials[2] $q_1(x) = a_0 + x$ and $q_2(x) = b_0$, and told that $d > M = b_0^{\alpha_0}$ divides both $q_1(x_0)$ and $q_2(x_0)$ for some $|x_0| < X = b_0^{\beta_0}$. As in the previous section we will work out the conditions on β_0 (in terms of α_0) for us to be assured of finding such common divisors d.

Rather than just consider $q_1(x)$ and $q_2(x)$ directly we will calculate $r(x)$ by considering the polynomials $p_i(x) = q_1(x)^{u-i}q_2(x)^i$ for some fixed integer u and $i = 0 \dots u$. Let h be the degree of the polynomial $r(x)$ we are trying to produce. We will see later how the optimal choice of h is related to u, but for the moment just notice that $p_i(x_0) = 0 \bmod d^u$ for $i = 0 \dots u$.

In order to find the desired polynomial $r(x)$ and to work out the size of its coefficients we must describe the lattice we build. To this end we show an example below with $h = 4$ and $u = 2$.

$$\begin{pmatrix} b_0^2 & 0 & 0 & 0 & 0 \\ b_0a_0 & b_0X & 0 & 0 & 0 \\ a_0^2 & 2a_0X & X^2 & 0 & 0 \\ 0 & a_0^2X & 2a_0X^2 & X^3 & 0 \\ 0 & 0 & a_0^2X^2 & 2a_0X^3 & X^4 \end{pmatrix}$$

The reason that we build the lattice generated by the rows of this type of matrix, is that each of the first $u+1$ rows corresponds to one of the polynomials $p_i(x)$, where the coefficient of x^j is placed into the $(j+1)^{\text{th}}$ column multiplied with $X^j = b_0^{j\beta_0}$. Notice that with this representation the integer matrix operations respect polynomial addition.

The remaining $h-u-1$ rows effectively allow for a *polynomial* multiple of the relation $(a_0+x)^u$ (polynomial multiples of the other relations are ignored because

[2] Notice that $q_2(x)$ is simply a constant, but we still refer to it as a (constant) polynomial.

of their obvious linear dependence). If these choices seem a little "plucked from the air", their justification can be seen in the determinant analysis.

We plan to reduce this lattice to obtain a small vector $\mathbf{r} = (r_0, r_1X, \ldots, r_hX^h)$ from which we obtain the polynomial $r(x) = \sum r_i x^i$.

It remains to see how small the entries of $\mathbf{r}$ will be. The dimension of the lattice is clearly $(h+1)$ and the determinant of the lattice can be seen to be

$$\Delta = X^{h(h+1)/2} b_0^{u(u+1)/2} = b_0^{u(u+1)/2+\beta_0 h(h+1)/2},$$

which means we are assured of finding a vector $\mathbf{r}$ such that $|\mathbf{r}| < c\Delta^{1/(h+1)}$ for some c which (as in the previous section) is asymptotically small enough (compared to $\Delta^{1/(h+1)}$) for us to ignore in the following analysis. To actually find $\mathbf{r}$ one would use the LLL algorithm (see [8]) or one of its variants (e.g. [12]).

Notice that, as mentioed above, this bound on $\mathbf{r}$ is also (approximately) a bound on the integers $|r(x')|$, where $r(x) = \sum r_i x^i$, and x' is any integer such that $|x'| < X$.

We therefore wish $\Delta^{1/(h+1)}$ to be less than $d^u > b_0^{\alpha_0 u}$, so that the roots of $r(x) = 0$ over the integers must contain all solutions x_0 such that $d > b_0^{\alpha_0}$ divides both $q_1(x_0)$ and $q_2(x_0)$, and $|x_0| < X$.

For this to happen we require that $(u(u+1) + \beta_0 h(h+1))/(2(h+1)) < \alpha_0 u$, i.e.

$$\beta_0 < \frac{u(2(h+1)\alpha_0 - (u+1))}{h(h+1)}.$$

For a given α_0 the optimum choice of h turns out to be at approximately u/α_0; in which case we obtain that whenever

$$\beta_0 < \alpha_0^2 - \frac{\alpha_0(1-\alpha_0)}{h+1},$$

we will find all the possible x_0.

This proves the existence of algorithm `GACD_L`. One simply reduces the relevant lattice with $h = \lceil(\alpha_0(1-\alpha_0)/\varepsilon\rceil - 1$ and $u = \lceil h\alpha_0 \rceil$, and finds the roots of the resulting polyomial equation over the integers. The fact that there are only a polynomial number of possible d follows from the fact that the number of x_0 is bounded by h.

4 Using Lattices to Solve GACDP

In this section we attempt to solve GACDP by recovering the x and y such that "a paricularly large" d divides both $a_0 + x$ and $b_0 + y$. As mentioned in section 2 there are exponentially many x and y for $d > b_0^{2/3}$ which implies that our lattice technique cannot work above this bound. However we will show that the method does work (heuristically) right up to this bound.

The approach taken is similar to that described in section 3, which we will make frequent reference to, but the analysis must now deal with bivariate polynomials.

A particularly interesting question is whether one can make what follows completely rigorous. Such an argument is presently evading author, and as shown in [9] one cannot hope for a argument for a general bivariate modular equation, but it does not rule out that in in this case (and other specific cases) it may be possible to *prove* what follows.

In the GACDP we are given two polynomials[3] $q_1(x,y) = a_0 + x$ and $q_2(x,y) = b_0 + y$, where $a_0 \sim b_0$, and told that $d > M = b_0^{\alpha}$ divides $q_1(x_0,y_0)$ and $q_2(x_0,y_0)$ for some $|x_0|, |y_0| < X = b_0^{\beta}$.

The first thing to notice is that we can extend the general approach of the last section easily, since if we have m polynomials $p_i(x,y) \in \mathbf{Z}[x,y]$, and we are assured that for some integers x_0, y_0 we have $p_i(x_0,y_0) = 0 \bmod t$ for all $i = 1 \ldots m$, then any (polynomial) linear combination of these polynomials, i.e. $r(x,y) = \sum_{i=1}^{m} u_i(x,y)p_i(x,y)$ also has the property that $r(x_0,y_0) = 0 \bmod t$.

Again rather than just considering $q_1(x,y)$ and $q_2(x,y)$ directly we will calculate $r(x,y)$ by considering the polynomials $p_i(x,y) = q_1(x,y)^{u-i}q_2(x,y)^i$ for some fixed integer u. In this new case the role of the variable h is as a bound on the total degree of the polynomials. Notice that we still have $p_i(x_0,y_0) = 0 \bmod d^u$.

As an example of the lattice we create we show below the matrix whose rows generate the lattice, for $h = 4$ and $u = 2$. Note that we use the symbol Y to denote bound on $|y_0|$, though in fact $Y = X = b_0^{\beta_0}$ in our case (this is done so that the entries of the matrix may be more easily understood).

$$\begin{pmatrix}
a_0^2 & 2a_0X & X^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_0^2X & 2a_0X^2 & X^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & a_0^2X^2 & 2a_0X^3 & X^4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_0b_0 & b_0X & 0 & 0 & 0 & a_0Y & XY & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_0b_0X & b_0X^2 & 0 & 0 & 0 & a_0XY & X^2Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & a_0b_0X^2 & b_0X^3 & 0 & 0 & 0 & a_0X^2Y & X^3Y & 0 & 0 & 0 & 0 & 0 & 0 \\
b_0^2 & 0 & 0 & 0 & 0 & 2b_0Y & 0 & 0 & 0 & Y^2 & 0 & 0 & 0 & 0 & 0 \\
0 & b_0^2X & 0 & 0 & 0 & 0 & 2b_0XY & 0 & 0 & 0 & XY^2 & 0 & 0 & 0 & 0 \\
0 & 0 & b_0^2X^2 & 0 & 0 & 0 & 0 & 2b_0X^2Y & 0 & 0 & 0 & X^2Y^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & b_0^2Y & 0 & 0 & 0 & 2b_0Y^2 & 0 & 0 & Y^3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & b_0^2XY & 0 & 0 & 0 & 2b_0XY^2 & 0 & 0 & XY^3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_0^2Y^2 & 0 & 0 & 2b_0Y^3 & 0 & Y^4
\end{pmatrix}$$

Again, since we are proving the bounds given in algorithm `GACD_L`, we must treat β_0 as a variable for the time being.

As can be seen the situation is slightly different to that previously obtained in section 3, mainly due to the differences in the linear dependencies of polynomial multiples of the $p_i(x,y)$. As we see we can multiply each $p_i(x,y)$ by x^j for all $j = 0 \ldots h-u$ now, without incurring any linear dependencies. The down side is that only $(h-u)(h-u+1)/2$ other rows corresponding to multiples of $(b_0+y)^u$ may be added.

Alternatively seen, one can think of the $p_i(x,y)$ as $(a_0+x)^s(b_0+y)^t$ for $u \leq s+t \leq h$, since these generate the same basis as the above ones. We consider these because they admit an easier analysis, though in practice we would probably reduce the above ones because they are slightly more orthogonal.

[3] Notice that $q_1(x,y)$ is not dependent on y, and $q_2(x,y)$ is not dependent on x, but we cannot keep the univariate nature of them for long.

Again we wish to reduce this lattice, and recover a polynomial $r_1(x, y)$ which is small for all $|x|, |y| < X$. However knowing that all the x, y solutions to the GACDP are solutions to $r_1(x, y) = 0$ does not assure us of finding the x_0, y_0 we require; in general we cannot solve a bivariate equation over the integers.

The common (heuristic) argument at this stage, is to find another small vector in the basis, from which one produces another polynomial $r_2(x, y) = 0$ over the integers. If we are fortunate in that $r_1(x, y)$ and $r_2(x, y)$ are algebraically independent, then this does give us enough information to solve for all roots (x, y), e.g. using resultant techniques. Unfortunately we cannot *prove* that in general we can always find two small vectors which give rise to *algebraically* independent polyomials, although this would seem likely, and indeed is borne out by practical results (see Table 1).

Assuming that we can find two algebraically independent polynomials, we still have the problem of working out what bounds this lattice method implies. Working out the determinant of a lattice given by a non-square matrix can be a major piece of work; see for example [3]. Fortunately there is a trick in our example, which may prove useful in more general situations.

It is well known that the determinant of the lattice is bounded above by the product of all the norms of the rows. Our strategy for analysing the determinant of this lattice will be to perform some non-integral row operations[4] and then use the product of the norms of the rows.

As an example of this if we consider the fourth row of the above lattice, then this clearly corresponds to the polynomial $(a_0 + x)(b_0 + y)$. By writing $(b_0 + y)$ as $y_0 - (b_0/a_0)x + (b_0/a_0)(a_0 + x)$ we see that

$$(a_0 + x)(b_0 + y) = (a_0 + x)(y - \frac{a_0}{b_0}x) + \frac{a_0}{b_0}(n + x)^2.$$

This means that taking a_0/b_0 times the first row, away from the fourth row, will leave us with a polynomial representing $(a_0 + x)(y - (a_0/b_0)x)$. Since $a_0 \sim b_0$, if we look at each of the entries of the vector corresponding to this polynomial then we see that they are each less than b_0X (since $a_0 < b_0$), so this a bound for the norm of the vector too (again we ignore the aymptotically small factor of the difference of these two norms). This bound is a vast improvement on the b_0^2 we may have naïvely bounded the vector by.

In a similar way, we also see that the norms of the vectors corresponding to the polynomials $(a_0 + x)^s$ can be bounded by $X^{s-u}b_0^u$, since we can subtract suitable multiples of the vectors corresponding to the polynomials $(a_0 + x)^i$ for $i < s$.

In general the vector corresponding to a polynomial $p_i(x, y) = (a_0 + x)^s(b_0 + y)^t$ will contribute $X^{s+t-u}Y^t b_0^{u-t}$ for $0 \leq t \leq u$ and X^sY^t for $u < t \leq h$. We can write the contribution of the polynomials in a table indexed by s and t, e.g. for the above example with $h = 4$ and $u = 2$

[4] These non-integral row operations will mean that the resulting matrix will no longer be a basis for our lattice, but it will have the same determinant which is all we care about estimating.

Y^4				
Y^3	XY^3			
Y^2	XY^2	X^2Y^2		
	b_0Y	b_0XY	b_0X^2Y	
		b_0^2	b_0^2X	$b_0^2X^2$

This representation allows us to count the dimension of the lattice and its determinant easily. The dimension is clearly $m = (h+1-u)(h+u+2)/2$, and the determinant is

$$\Delta = Y^{(h+1-u)(h(h+u+2)+u(u+1))/6} b_0^{u(u+1)(h+1-u)/2} X^{(h-u)(h+1-u)(h+2u+2)/6} = b_0^{\delta}$$

where $\delta = u(u+1)(h+1-u)/2 + \beta_0(h+1-u)(2h^2+4h+2uh-u^2-u)/6$.

As in section 3 we require that $\delta < mu\alpha_0$, so that the vectors LLL finds[5] will be valid over the integers. Representing this in terms of β_0 we obtain

$$\begin{aligned}\beta_0 &< \frac{3u(h+u+2)\alpha_0 - 3u(u+1)}{2h(h+u+2)-u(u+1)} \\ &= \frac{3u(h+u)\alpha_0 - 3u^2}{2h(h+u)-u^2} - \frac{3u(h-u)(2h-u\alpha_0)}{(2h(h+u)-u^2)(2h(h+u+2)-u(u+1))}.\end{aligned}$$

The optimal setting for γ such that $h = u/\gamma$ is now

$$\gamma = \frac{2-2\alpha_0-\sqrt{4-4\alpha_0-2\alpha_0^2}}{3\alpha_0-2}$$

for which we obtain the bound

$$\beta_0 < 1 - \frac{1}{2}\alpha_0 - \sqrt{1-\alpha_0-\frac{1}{2}\alpha_0^2} - \varepsilon(h,\alpha_0)$$

where for completeness we give the precise $\varepsilon(h,\alpha_0) = \varepsilon_1/\varepsilon_2$; namely

$$\begin{aligned}\varepsilon_1 &= 3(3\alpha_0-2)\Big((\alpha_0-1)(11\alpha_0^2+8\alpha_0-12)\sqrt{4-4\alpha_0-2\alpha_0^2} \\ &\quad +(\alpha_0^2-16\alpha_0+12)(\alpha_0^2+2\alpha_0-2)\Big) \\ \varepsilon_2 &= 4h(\alpha_0^2+2\alpha_0-2)\left((10\alpha_0-8)\sqrt{4-4\alpha_0-2\alpha_0^2}+23\alpha_0^2-44\alpha_0+20\right) \\ &\quad +2(3\alpha_0-2)\Big((34\alpha_0^2-55\alpha_0+22)\sqrt{4-4\alpha_0-2\alpha_0^2} \\ &\quad +(19\alpha_0-14)(\alpha_0^2+2\alpha_0-2)\Big).\end{aligned}$$

Notice that $\lim_{h\to\infty}\varepsilon(h,\alpha_0) = 0$, and so this proves the existence of the algorithm `GACD_L`.

[5] Actually there are a few minor errors that can creep in due to: underestimating the determinant (since $a_0 \sim b_0$, but $a_0 \neq b_0$), transferring between the L_1 norm and L_2 norms, the fact that there is a $2^{(m-1)/4}$ LLL approximation factor and the common divisor may be overestimated (again since $a_0 \sim b_0$, but $a_0 \neq b_0$).

5 An Equivalent Problem?

In this paper we have been considering equations where d divides both $a_0 + x$ and $b_0 + y$, and d is particularly large with respect to the sizes of x and y. Notice that we may write this as

$$b'(a_0 + x) - a'(b_0 + y) = 0$$

where now $a' = (a_0 + x)/d$ and $b' = (b_0 + y)/d$ are particularly small.

A very related problem to this is finding small a' and b' such that there exists small x, y which satisfy

$$b'(a_0 + x) - a'(b_0 + y) = 1.$$

The author knows of no *reduction* between these two problems, but as we shall the techniques used to solve them are remarkably similar.

The *partial* variant of the second problem is very related to cryptography; indeed it has recently been named “the small inverse problem” in [3]. The connection with cryptography was first realised in [15] where it was shown (via a continued fraction method) that the use of a low decrypting exponent in RSA (i.e. one less than a quarter of the bits of the modulus N) is highly insecure; the public information of N and e effectively leaks the factorisation of N in polynomial time!

The problem was re-addressed in [3], where the new attack was now based on Coppersmith's lattice techniques, and the bound for which the decrypting exponent likely reveals the factorisation of N (in polynomial time) was increased to $1 - 1/\sqrt{2}$ of the bits of N. It is a heuristic method, but seems to work very reliably in practice.

In [3] it was hypothesised that this bound might be increased to $1/2$ the bits of N, because the bound of $1 - 1/\sqrt{2}$ seemed unnatural. However the author would like to point out that this state of affairs is unlikely, and in fact the bound of $1 - 1/\sqrt{2}$ *is* natural in that it is exactly what one would expect from the related PACDP.

Indeed if one had the bound $\beta_0 = 1/2 - \varepsilon$ for $\alpha_0 = 1/2$ for the PACDP (as one might expect the two problems to have the same bound), then this would imply a polynomial time factoring algorithm for RSA moduli[6]!

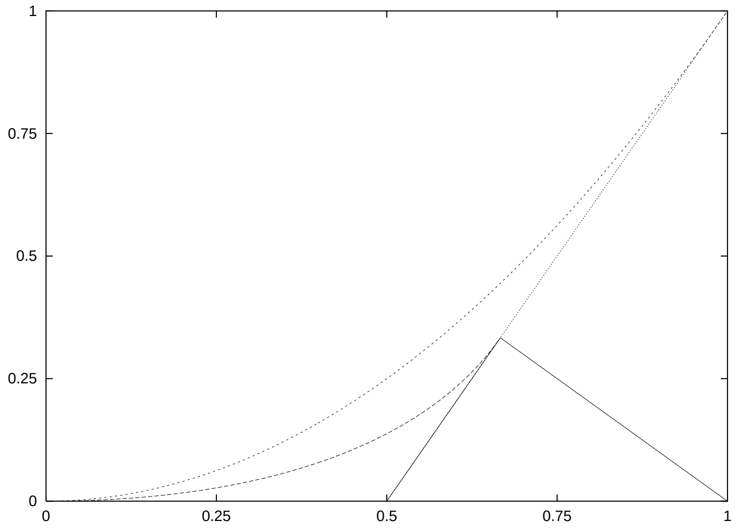
We note that this does not *prove* the falsity of the conjecture (even assuming factoring is not possible in polynomial time) because there is no known reduction between the small inverse problem and the PACDP (in either direction). However it does mean that for the conjecture to be true, the fact that the r.h.s. of the above equation is 1 rather than 0 must be of key importance to the algorithm which finds a', b' (which is not the case in any of the known attacks).

[6] One would just need to guess a constant amount of the top order bits of a factor of p, and use the algorithm to find the remaining bits.

6 Results

We sum up the results in the last sections by showing their bounds graphically. The variables $\alpha = \log_{b_0} M$ and $\beta = \log_{b_0} X$ are represented below by the $x-$ and $y-$ axes respectively. The lines show the maximum β for which the common divisor d can still be recovered for any given $\alpha \in (0 \ldots 1)$.

Figure 61. *The Bounds Implied by the Approximate Common Divisor Algorithms.*



The top line is $\beta = \alpha^2$ which is the bound implied by `PACD_L`. It is responsible for theorems such as "factoring with high bits known", and (with a slight extension) "factoring integers of the form $N = p^r q$"; see [1] and [5].

The straight line given by $\beta = 2\alpha - 1$ is the bound implied by `PACD_CF`, and is equivalent to Wiener's attack on low decrypting exponent RSA. The line for `GACD_CF` starts off the same, but after $\alpha = 2/3$ it then becomes the line $\beta = 1 - \alpha$ to ensure a polynomial sized output.

The curved line between the previous two is

$$\beta = 1 - \frac{1}{2}\alpha - \sqrt{1 - \alpha - \frac{1}{2}\alpha^2}.$$

This is a new (but heuristic) lattice based result, explained in section 4, which also applies to GACDP. It interestingly shows that the problem may be solved even when $\alpha < 1/2$.

We now give some practical results to demonstrate the last of these methods working in practice. We show that indeed we can recover the common divisor d

in reasonable time, and that the size of the short vectors are not significantly smaller than predicted by the Minkowski bound, which impies that no sublattice of the one we reduce will reveal a better determinant analysis (this ratio is given logarithmically below by δ).

The results were carried out on a Pentium II 700MHz machine, running Redhat Linux. The software was written in `C++` and used the NTL Library [13]. Notice that for consistency the total number of bits of a and b was kept near 1024.

Table 1. Table of Practical Results.

# bits of divisor	# bits of a,b	α	β_{max}	h	u	$time(s)$	δ	# bits of error	β
205	1025	0.2	0.016	10	1	731	0.941	5	0.004
307	1023	0.3	0.041	5	1	8	0.998	15	0.012
				6	1	20	0.972	20	0.019
				7	1	43	0.965	22	0.019
410	1025	0.4	0.079	4	1	2	0.984	43	0.041
				5	2	28	0.990	41	0.040
				6	2	83	0.987	51	0.049
				7	2	197	0.990	56	0.055
512	1024	0.5	0.137	4	2	9	0.994	103	0.100
				5	2	34	0.993	107	0.104
				6	3	261	0.996	113	0.110
614	1023	0.6	0.231	4	3	18	0.998	213	0.208
				5	3	100	0.990	207	0.204
				6	4	507	0.997	216	0.211

(We bring to the readers attention the fact that some of the above choices for u are sub-optimal. This effect can be noticed in the bound on β).

7 Conclusions and Open Problems

The first problem is to find more uses for approximate integer common divisor problems in cryptography, or any other part of computational mathematics (especially for the general case). Without this the techniques described herein will find it hard to reach a large target audience.

Another interesting question is to see if one can find reductions between the "= 0" and "= 1" variants given in section 5, i.e. the small inverse problem and PACDP.

The next obvious thing to do is to generalise ACDPs even more. For instance one could complete the analysis (aluded to in section 4) of when d^r divides a number close to one of the inputs. Alternatively one could work out the bounds when one has many inputs, all close to numbers which share a large common divisor d. For completeness these extensions will appear in the final version of this paper.

The last problems are of a more fundamental nature to do with lattice analysis. The proof of algebraic independence of the two bivariate polynomials still remains as a key result to prove in this field. A proof would turn both this method and the ones described in [3] and [4] in to "fully fledged" polynomial time algorithms, rather than the heuristic methods they are currently written up as. Of course as shown by [9] it is not possible to hope for a general solution to bivariate modular equations, but in this particular case (and others) one may hope to find a rigorous proof.

Finally we ask if the bounds implied by section 4 are the best possible, in polynomial time? One way in which this result may not be optimal, is that our determinant analysis may be too pessimistic, or alternatively there may be a sublattice of this lattice, for which the reduced dimension and determinant imply better bounds on the GACDP (for example this type of effect was seen in the first lattice built in [3]).

Evidence against both of these states of affairs is given by the results in Table 1, i.e. they do show that the vectors obtained are approximately what one would expect from the Minkowski bound (though for smaller α there is a small observed discrepancy), which means that neither of these above situations is likely with our method.

This being said, it still remains a very interesting and open theoretical question to be able to identify, in advance, which lattices are built "incorrectly", i.e. those for which some of the rows actually hinder the determinant analysis (e.g. like the first one in [3]).

A third way to improve the results is that one could build an entirely different lattice, whose analyis simply admitted better bounds than the ones reached here. Nothing is presently known about the existence or non-existence of such an algorithm.

Acknowledgements

The author would particularly like to thank Don Coppersmith, Yuval Ishai and Phong Nguyen for interesting conversations relating to this subject, and is also very grateful for the period of time spent on this problem at HP Labs in Bristol.

References

1. D. Coppersmith. Finding a small root of a bivariate integer equation *Proc. of Eurocrypt'96* Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, 1996
2. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society (AMS)* Vol. 46, No. 2, pp. 203–213, 1999.
3. D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$ *IEEE Transactions on Information Theory,* Vol 46, No. 4, pp. 1339–1349, July 2000.

4. D. Boneh, G. Durfee and Y. Frankel. An attack on RSA given a small fraction of the private key bits. *In proceedings AsiaCrypt'98*, Lecture Notes in Computer Science, Vol. 1514, Springer-Verlag, pp. 25–34, 1998.
5. D. Boneh, G. Durfee and N. Howgrave-Graham Factoring $N = p^r q$ for large r. *In Proceedings Crypto '99*, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 326–337, 1999.
6. G.H. Hardy and E.M. Wright. An introduction to the theory of numbers, 5'th edition. Oxford University press, 1979.
7. N.A. Howgrave-Graham. Computational mathematics inspired by RSA. Ph.D. Thesis, Bath University, 1999.
8. A.K. Lenstra, H.W. Lenstra and L. Lovász. Factoring polynomials with integer coefficients *Mathematische Annalen*, Vol. 261, pp. 513–534, 1982.
9. K.L. Manders and L.M. Adleman. NP-Complete decision problems for binary quadratics *JCSS* Vol. 16(2), pp. 168–184, 1978.
10. P. Nguyen and J. Stern. Lattice reduction in cryptology: An update", *Algorithmic Number Theory – Proc. of ANTS-IV*, volume 1838 of LNCS. Springer-Verlag, 2000.
11. T. Okamoto. Fast public–key cryptosystem using congruent polynomial equations *Electronic letters*, Vol. 22, No. 11, pp. 581–582, 1986.
12. C–P. Schnorr. A hierarchy of polynomial time lattice bases reduction algorithms *Theoretical computer science*, Vol. 53, pp. 201–224, 1987.
13. V. Shoup. NTL: A Library for doing Number Theory (version 4.2) `http://www.shoup.net`
14. B. Vallée, M. Girault and P. Toffin. *Proceedings of Eurocrypt '88* LNCS vol. 330, pp. 281–291, 1988.
15. M. Wiener. Cryptanalysis of short RSA secret exponents *IEEE Transactions of Information Theory* volume **36**, pages 553-558, 1990.