LINEAR CONGRUENTIAL GENERATORS DO NOT PRODUCE RANDOM SEQUENCES

A.M. Frieze[*], R. Kannan[**] and J.C. Lagarias[***]

[*]GSIA, Carnegie-Mellon University and Queen Mary College, London,
[**]Computer Science Department, Carnegie-Mellon University,
[***]AT&T Bell Laboratories, Murray Hill.

Abstract

One of the most popular and fast methods of generating "random" sequence are linear congruential generators. This paper discusses the predictability of the sequence given only a constant proportion α of the leading bits of the first few numbers generated. We show that the rest of the sequence is predictable in polynomial time, almost always, provided $\alpha > 2/5$.

One of the most popular and fast methods of generating "random" sequences are linear congruential generators. These work as follows: a modulas M, a multiplier a relatively prime to M and an increment c are picked. Then starting at a random "seed" X_1 one generates the sequence $\{X_i\}$ given by

$$X_{i+1} = a \cdot X_i + c (\text{mod } M) \qquad (0)$$

(Thus the X_i are all integers between 0 and M - 1.) Knuth (Vol. 2) contains an elaborate discussion of linear congruential generators (LCG). The sequences produced by LCG's have been shown to satisfy various statistical tests of randomness for proper choices of the modulas and multiplier. (Knuth-Vol. 2). However it does not immediately follow from these that these sequences are "unpredictable" - which one would intuitively expect a random sequence to be. This aspect of randomness has been formalized by cryptographers Shamir (1980), Blum and Micali (1982), Yao (1982) and Goldreich, Goldwasser and Micali (1984). Also, the thesis that problems that can be done in random polynomial time are essentially tractable is based on the hypothesis that a deterministic polynomial time bounded process can produce sequences that are indistinguishable from truly random sequences in deterministic polynomial time. (See Cook (1983) for a discussion of this thesis). Indeed the general observation so far seems to be that probabilistic (coin-tossing) algorithms work well in practice. In view of this it is important to analyse one of the most popular random number generators - the linear congruential generator for predictability.

It has been suggested (Knuth 1980) that a way of producing secure sequences from an LCG is to output the leading part of each of the X_i's - say the leading half of the bits.† The main result of this paper is to show that this sequence is not secure. Knuth (1980), Plumstead (1982) and Reeds (1977) have considered the question of whether bits generated by linear congruential generators are predictable. Plumstead (1982) uses a clever idea to show that if all the bits of several consecutive X_i's are known, then the multiplier a can be inferred and with greater difficulty the modulas too, thus demonstrating that when all bits of X_i are announced, the sequence becomes predictable even if the modulas and multiplier are unknown. Knuth (1980) considers the problem when

†Note that if the modulas is known it is certainly insecure to output X_i, X_{i+1}, X_{i+2} for any i, for then a is given by $(X_{i+1}-X_i)^{-1} \cdot (X_{i+2}-X_{i+1})$ and thence c can be found. Here the inverse is modulo M, if the inverse does not exist, a simple modification of the expression suffices to find a.

the multiplier and modulas are unknown and only a small fraction of the bits of several consecutive X_i's are announced. For this case, he devises an exponential time algorithm to infer the hidden information. Reeds (1977) considers some special cases with fixed multipliers. Plumstead (1982) also treats the case when the trailing $O(\log(n))$ bits of several consecutive X_i's are unknown.

To describe our result we first introduce some notation: let $n = 2m$ be the number of bits in M. We break X_i into two equal parts:

$$X_i = 2^m \cdot y_i + z_i \tag{1}$$

where $0 \le y_i, z_i \le 2^m$. The problem we consider is: given M, a, c, $y_1, y_2, y_3, \dots y_\ell$ for some ℓ, can one determine z_1 (and then of course all the X_i can be easily computed.) The main result is an algorithm A with the following properties:

1) A is deterministic polynomial time bounded. Indeed A runs in time $O(n^2 \log n \log\log\log n)$.

2) It takes as input integers M, a and integers y_1, y_2 and y_3, $0 \le y_1, y_2, y_3 \le 2^m$ and returns an integer z_1 between 0 and 2^m or returns the answer "cannot solve the instance". (See (3) below)

3) For each M, there is a set S_M containing at least $(1 - O(M^{-(1/5)})$ of the integers modulo M such that

a) for any a in S_M, and any c, given y_1, y_2, y_3 integers in $[0, \sqrt{M}]$, there is a unique z_1, z_2 and z_3 in $[0, \sqrt{M}]$ such that x_1, x_2 and x_3 defined by (1) satisfy (0).

b) there is a polynomial-time algorithm that given a,M tests whether a is in S_M.

c) whenever $a \in S_M$, the algorithm A gives the correct (unique) answer; if $a \notin S_M$, A returns "cannot solve".

Algorithm A

We use the algorithm of Kannan (1983) to find an integer solution to

$$\begin{aligned} az_1 - z_2 + Mp_1 &= Y_1 \\ az_2 - z_3 + Mp_2 &= Y_2 \\ 0 \le z_i \le 2^m & \qquad i = 1, 2, 3. \end{aligned} \tag{2}$$

where $Y_i = 2^m(y_{i+1} - ay_i - c) \pmod M$ for $i = 1,2$ and p_1, p_2 are new integer variables. (We remark that Lenstra's (1979) algorithm could take $\Omega(n^9)$ time).

Now clearly if z_1, z_2, z_3 are the "hidden bits" of an LCG then they will form a solution to (2) with suitable values for p_1, p_2. The key issue is whether or not there are any other solutions. If there are none then our method is valid.

We define the set S_M for which we know that the solution is unique.

Suppose that there is another solution $(z_1', z_2', z_3', p_1', p_2')$ to (2). Then putting $u_i = z_i - z_i'$ for i = 1, 2, 3 we have

$$\begin{aligned} u_2 &= au_1 \quad (\mathrm{Mod}\ M) \\ u_3 &= au_2 \quad (\mathrm{Mod}\ M) \\ |u_i| &\le 2^{m+1} \qquad i = 1,2,3 \end{aligned} \tag{3}$$

where we define (Mod M), as opposed to (mod M) to be the least absolute value residue i.e. $-M/2 \le y(\mathrm{Mod}\ M) < M/2$. We can assume without loss of generality that $u_1 > 0$ (clearly if $u_1 < 0$ we replace u_1 by $-u_1$. If $u_1 = 0$ we find that $u_2 = u_3 = 0$ as $|u_i| < M$. But then $p_i = p_i'$ for $i = 1,2$ follows easily and our solutions are not distinct.)

Thus if

$$B_M = \{0 \le a \le M-1 : \exists x, 0 < x \le 2^{m+1} \text{ such that } |a^i x\ (\mathrm{Mod}\ M)| \le 2^{m+1},\ i=1,2\}$$

and

$$S_M = \{0, 1, \dots M-1\} - B_M$$

then we have

<u>If</u> $a \in S_M$ <u>then there is at most one solution to (2) and our algorithm finds it.</u> (4)

Our next task is to bound the size of B_M. For $0 < x \le L = 2^{m+1}$ let

$B(x) = \{0 \le a \le M-1: |ax \pmod M|, |a^2x \pmod M| \le L\}$.

Now

$$|B_M| \le \sum_{x=1}^{L} |B(x)| \qquad (5)$$

as each $a \in B_M$ is counted at least once in the sum on the right hand side of (5).

Consider now a fixed x, $0 < x \le L$ and assume first that x and M are relatively prime. Let $w = x^{-1} \pmod M$.

Then putting $y = ax \pmod M$ and using $a^2x = wy^2 \pmod M$ we obtain

$$|B(x)| = |X_w| \qquad (6)$$

where $X_w = \{-L \le y \le L: |wy^2 \pmod M| \le L\}$.

We now obtain a bound for the size of $|X_w|$ which will be used with (5) and (6) to bound $|B_M|$.

Consider the function ϕ: $X_w^2 \rightarrow Z$ defined by

$$\phi(y_1, y_2) = w(y_1^2 + y_2^2) \pmod M. \qquad (7)$$

Note that

$$|\phi(y_1, y_2)| \le 2L \qquad \text{for } y_1, y_2 \in X_w. \qquad (8)$$

Let now $\varepsilon > 0$ be an arbitrarily small positive real number. We show that there exists a_ε such that if $|u| \le 2L$ then

$$|\phi^{-1}(u)| \le a_\varepsilon \underbrace{L^{2+2\varepsilon}/M} \qquad (9)$$

$\Downarrow$

$|X_w|^2 \le 2L \times$

To see this consider a fixed $(y_1,y_2) \in \phi^{-1}(u)$ having the smallest value of $y_1^2 + y_2^2$. Then $(y_1', y_2') \in \phi^{-1}(u)$

if and only if

$$w(y_1'^2 + y_2'^2) = w(y_1^2 + y_2^2) \pmod M$$

if and only if

$$y_1'^2 + y_2'^2 = y_1^2 + y_2^2 \pmod M$$

if and only if

$$y_1'^2 + y_2'^2 = y_1^2 + y_2^2 + pM$$

for some integer p, $0 \le p \le \bar p = \lfloor (2L^2 - y_1^2 - y_2^2)/M \rfloor$.

Now for non-negative integer n, let $\psi(n)$ denote the number of distinct integer solutions (x,y) to the equation

$$x^2 + y^2 = n.$$

It follows that

$$|\phi^{-1}(u)| \le \sum_{p=0}^{\bar p} \psi(y_1^2 + y_2^2 + pM) \qquad (10)$$

Now it is known (Le Veque (1956) for example) that for any $\varepsilon > 0$ there exists b_ε such that $\psi(n) \le b_\varepsilon n^\varepsilon$. It follows from (10) that (9) holds with $a_\varepsilon = 2^{1+\varepsilon} b_\varepsilon$.

It then follows from (8) and (9) that

$$|X_w| \le (4a_\varepsilon L^{3+2\varepsilon}/M)^{1/2} \qquad (11)$$

which completes the case for x and M relatively prime.

If $d=d(x) = \gcd(x,M)>1$ we find that

$|B(x)| \overset{*}{=} d(x)\ |\{-\hat L \le y \le \hat L : |\hat\omega y^2 \pmod{\hat M}| \le \hat L\}|$
where $\hat M = M/d$, $\hat L = |L/d|$ and $\hat\omega = (x/d)^{-1} \pmod{\hat M}$.

$a \in B(x) \Rightarrow a + i\hat M \in B(x)$, $i = 0, 1, \cdots d-1$

It follows from (11) that $|B(x)| \leq d(x) (4\hat{a}_\varepsilon L^{3+2\varepsilon}/\hat{M})^{1/2}$ and hence that

$$|B_M| \leq \sum_{x=1}^{L} d(x)^{-\varepsilon}(4a_\varepsilon L^{3+2\varepsilon}/M)^{1/2} \qquad (12)$$

$$\leq c_\varepsilon (L^{5+2\varepsilon}/M)^{1/2}$$

where $c_\varepsilon = 2a_\varepsilon^{1/2}$.

Substituting $L = 2^{m+1}$ and putting $\varepsilon = 1/20$ yields $|B_M| = O(M^{4/5})$ as stated.

We note that if we are given slightly fewer than n/2 bits i.e. $\lfloor \alpha n \rfloor$ bits where $\alpha > 2/5$ then simply putting $m = \lfloor (1-\alpha)n \rfloor$ in the above analysis shows that our method works except on a set of a's of size $O(M^{2-5\alpha/2+\varepsilon})$ for any $\varepsilon > 0$.

We now consider the problem of testing for $a \in B_M$. This is again an integer program in a fixed number of variables. Thus $a \in B_M$ if and only if there is a solution to

$$\begin{aligned} &1 \leq x \leq L \\ &-L \leq ax + p_1 M \leq L \\ &-L \leq a^2 x + p_2 M \leq L \\ &x,\ p_1,\ p_2 \text{ integer.} \end{aligned}$$

<u>Extensions</u> the problem naturally arises: what if instead of half the bits we are only given a much smaller fraction of them? Then, of course we may require portions of more than 3 of the X_i's, but will a fixed number depending only on α do? We show that the answer is affirmative provided the following number theory conjecture is true:

Corresponding to any fraction $\alpha \in (0,1)$ there exists a natural number ℓ and a fraction $\delta \in (0,1)$ such that the cardinality of the set $B_{\alpha,M}$ defined below is $O(M^\delta)$.

$$B_{\alpha,M} = \{a : 0 \leq a \leq M-1; \exists x, 0 < x \leq M^\alpha \text{ such that } |a^i x (\text{Mod } M)| \leq M^\alpha,\ i = 1,2,\ldots \ell\}.$$

We have proved the conjecture when M is square free. However the conjecture is open for general M.

We next consider the case where the constant c in (0) is not known. As it turns out, we can proceed in a similar manner to the above. This time we need the first 3 numbers generated. Using the decomposition (1) we will be looking for an <u>integer</u> solution to

$$\begin{aligned} az_1 - z_2 + c + Mp_1 &= Y_1 \\ az_2 - z_3 + c + Mp_2 &= Y_2 \\ az_3 - z_4 + c + Mp_3 &= Y_3 \end{aligned} \qquad (13)$$

$$-M < c < M$$
$$-2^m \leq z_i \leq 2^m \qquad i = 1,2,3,4$$

where $Y_i = 2^m(y_{i+1} - ay_i) \pmod M$ for i = 1,2,3. We show next that if we change the definition of B_M slightly by replacing 2^{m+1} by 2^{m+2} then

$$a \in S_M \text{ implies (13) has a unique solution} \qquad (14)$$

Suppose $(z_1', z_2', z_3', z_4', c', p_1', p_2', p_3')$ is an alternative solution. Put $v_i = z_i - z_i'$ for i = 1,2,3,4 and the $u_i = v_i - v_{i+1}$ for i = 1,2,3. It follows that (3) holds with 2^{m+1} replaced by 2^{m+2}.

Finally the case when a and possibly M are also unknown in addition to a fraction of the bits of X_i, remains an interesting open problem.

The ideas used in this paper will yield an algorithm for the case when M is odd and the trailing half of the bits are given to us. (When M is even these bits do not form a random sequence - this can be seen from basic considerations.) The sets B_M, S_M do not change.

<u>Acknowledgment</u> We thank Rick Statman for useful discussions and M. Blum and J. Plumstead for bringing the problem to our attention.

References

M. Blum and S. Micali, "How to generate cryptographically strong sequence of pseudo random bits?" Proceedings of the 23rd IEEE Symposium of the Foundations of Computer Science (1982).

S. Cook, "An overview of computational complexity" - 1982 ACM Turing Award lecture, Communications of the ACM Vol. 26, No. 6 June (1983) pp. 400-408.

O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions".

R. Kannan, "Improved algorithms for integer programming and related lattice problems" 15th Annual ACM symposium on theory of computing (1983) pp. 193-206.

D. E. Knuth, "Seminumerical algorithms. The art of computer programming" Vol. 2, Addison-Wesley (1969).

D. E. Knuth, "Deciphering a linear congruential encryption" Technical Report no. 024800, Stanford University (1980).

H. W. Lenstra, "Integer programming with a fixed number of variables" First announcement (1979) To appear in Mathematics of Operations research.

W. J. LeVegne, "Topics in number theory," Addison-Wesley, Mass. (1956).

J. Plumstead, "Inferring a sequence generated by a linear congruence" 23rd IEEE Symposium on the Foundations of Computer Science (1982), pp. 153-159.

J. Reeds, "Cracking a random number generator" Cryptologia, Vol. 1, Jan (1977).

A. Shamir, "On the generation of cryptographically strong pseudo random sequences" Seventh International Colloquium on Automate, Languages and Programming, (1980).

A. Yao, "Theory and applications of trapdoor functions" Proceedings of the 23rd IEEE Syposium of the Foundations of Computer Science (1982), pp. 80-91.

MATHEMATICS OF COMPUTATION
VOLUME 50, NUMBER 181
JANUARY 1988, PAGES 235-250

Polynomial Factorization and Nonrandomness of Bits of Algebraic and Some Transcendental Numbers

By R. Kannan, A. K. Lenstra, and L. Lovász

Abstract. We show that the binary expansions of algebraic numbers do not form secure pseudorandom sequences; given sufficiently many initial bits of an algebraic number, its minimal polynomial can be reconstructed, and therefore the further bits of the algebraic number can be computed. This also enables us to devise a simple algorithm to factor polynomials with rational coefficients. All algorithms work in polynomial time.

Introduction. Manuel Blum raised the following question: Suppose we are given an approximate root of an unknown polynomial with integral coefficients and a bound on the degree and size of the coefficients of the polynomial. Is it possible to infer the polynomial? We answer his question in the affirmative. We show that if a complex number α satisfies an irreducible polynomial $h(X)$ of degree d with integral coefficients in absolute value at most H, then given $O(d^2 + d \cdot \log H)$ bits of the binary expansion of the real and complex parts of α, we can find $h(X)$ in deterministic polynomial time (and then compute in polynomial time any further bits of α). Using the concept of secure pseudorandom sequences formulated by Shamir [23], Blum and Micali [3] and Yao [25], we then show that the binary (or m-ary for any m) expansions of algebraic numbers do not form secure sequences in a certain well-defined sense.

We are able to extend our results with the same techniques to transcendental numbers of the form $\log(\alpha), \cos^{-1}(\alpha)$, etc., where α is algebraic.

The technique is based on the lattice basis reduction algorithm from [16]. Our answer to Blum's question enables us to devise a simple polynomial-time algorithm to factor polynomials with rational coefficients: We find an approximate root of the polynomial and use our algorithm to find the irreducible polynomial satisfied by the exact root, which must then be a factor of the given polynomial. This is repeated until all the factors are found. This algorithm was found independently by Schönhage [22], and was already suggested in [16].

The technique of the paper also provides a natural, efficient method to compute with algebraic numbers.

This paper is the final journal version of [13], which contains essentially the entire contents of this paper.

1. A Polynomial-Time Algorithm for Blum's Question. Throughout this paper, **Z** denotes the set of the integers, **Q** the set of the rationals, **R** the set of the reals, and **C** the set of the complex numbers. The ring of polynomials with integral

Received December 23, 1985; revised October 13, 1986 and April 6, 1987.
1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 68Q15, 68Q25, 68Q40.

(complex) coefficients will be denoted $\mathbf{Z}[X]$ ($\mathbf{C}[X]$). The *content* of a polynomial $p(X)$ in $\mathbf{Z}[X]$ is the greatest common divisor (abbreviated gcd) of its coefficients. A polynomial in $\mathbf{Z}[X]$ is *primitive* if its content is 1. A polynomial $p(X)$ *has degree* d if $p(X) = \sum_{i=0}^{d} p_i X^i$ with $p_d \neq 0$. We write $\deg(p) = d$. The *length* $|p|$ of $p(X) = \sum_{i=0}^{d} p_i X^i$ is the Euclidean length of the vector $(p_0, p_1, \ldots, p_d)$; the *height* $|p|_\infty$ of $p(X)$ is the L_∞-norm of the vector $(p_0, p_1, \ldots, p_d)$, so $|p|_\infty = \max_{0 \le i \le d} |p_i|$. An *algebraic number* is a root of a polynomial with integral coefficients. The *minimal polynomial* of an algebraic number α is the irreducible polynomial in $\mathbf{Z}[X]$ satisfied by α. The minimal polynomial is unique up to units in $\mathbf{Z}$ (see, for example, [11]). The *degree* and *height* of an algebraic number are the degree and height, respectively, of its minimal polynomial. The real and complex parts of a complex number z will be denoted $\mathrm{Re}(z)$ and $\mathrm{Im}(z)$ respectively.

A *lattice* in $\mathbf{R}^n$ is a set of the form

$$\left\{ \sum_{i=1}^{k} \lambda_i b_i \colon \lambda_i \in \mathbf{Z} \right\},$$

where $b_1, b_2, \ldots, b_k$ are linearly independent vectors in $\mathbf{R}^n$. The lattice is said to be generated by the vectors $b_1, b_2, \ldots, b_k$, which form a *basis* for the lattice. The lattice is denoted $L(b_1, b_2, \ldots, b_k)$. An important result we need is the basis reduction algorithm from [16, Section 1]. We will only state the consequence of this algorithm used in this paper. Denote by $|\cdot|$ the ordinary Euclidean length on $\mathbf{R}^n$.

(1.1) THEOREM (cf. [16, Propositions (1.11) and (1.26)]). *Let*

$$L = L(b_1, b_2, \ldots, b_k)$$

be a lattice in $\mathbf{Z}^n$ *and let* $B \in \mathbf{R}$, $B \geq 2$, *be such that* $|b_i|^2 \leq B$ *for* $1 \leq i \leq k$. *It takes* $O(n \cdot k^3 \cdot \log B)$ *arithmetic operations (additions, subtractions, multiplications, and divisions) on integers having* $O(k \cdot \log B)$ *binary bits to transform the basis* $b_1, b_2, \ldots, b_k$ *by means of the basis reduction algorithm into a reduced basis* $v_1, v_2, \ldots, v_k$ *for* L. *The first vector* v_1 *in the reduced basis has length at most* $2^{(k-1)/2} \cdot \Lambda_1(L)$, *where* $\Lambda_1(L)$ *is the length of a shortest nonzero vector in* L.

Now we are ready to describe the idea behind our main result. Suppose upper bounds d and H on the degree and height, respectively, of an algebraic number α are known. Then we show that a sufficiently close rational approximation $\bar{\alpha}$ to α enables us to determine the minimal polynomial $h(X)$ of α.

Given $\bar{\alpha}$, we compute rational approximations $\bar{\alpha}_i$ to the powers α^i of α. For a polynomial $g = \sum_i g_i X^i \in \mathbf{C}[X]$ we introduce the following notation for the approximated evaluation of g at α:

$$g_{\bar{\alpha}} = \sum_i g_i \bar{\alpha}_i. \tag{1.2}$$

Suppose the degree of $h(X)$ is n, $n \leq d$. We try the values of $n = 1, 2, \ldots, d$ in order. With n fixed, we define for each positive integer s the lattice L_s in $\mathbf{R}^{n+3}$ generated by $b_0, b_1, \ldots, b_n$, which are the rows (in order) of the following $(n+1) \times (n+3)$

matrix:

$$(1.3)\qquad \begin{bmatrix} 1 & 0 & 0 & \cdot & \cdot & \cdot & 0 & 2^s\cdot\mathrm{Re}(\bar\alpha_0) & 2^s\cdot\mathrm{Im}(\bar\alpha_0) \\ 0 & 1 & 0 & \cdot & \cdot & \cdot & 0 & 2^s\cdot\mathrm{Re}(\bar\alpha_1) & 2^s\cdot\mathrm{Im}(\bar\alpha_1) \\ 0 & 0 & 1 & \cdot & \cdot & \cdot & 0 & 2^s\cdot\mathrm{Re}(\bar\alpha_2) & 2^s\cdot\mathrm{Im}(\bar\alpha_2) \\ \cdot & \cdot & \cdot & \cdot & & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & & & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & 1 & 2^s\cdot\mathrm{Re}(\bar\alpha_n) & 2^s\cdot\mathrm{Im}(\bar\alpha_n) \end{bmatrix}.$$

Corresponding to a polynomial $g = \sum_{i=0}^{n} g_i X^i$ in $\mathbf{Z}[X]$ of degree at most n (where some of the g_i are possibly zero), we have a vector $\tilde g$ in the lattice L_s defined by

$$(1.4)\qquad \tilde g = \sum_{i=0}^{n} g_i b_i.$$

Clearly,

$$|\tilde g|^2 = g_0^2 + g_1^2 + \cdots + g_n^2 + 2^{2s}\left(\mathrm{Re}\left(\sum_{i=0}^{n} g_i\bar\alpha_i\right)\right)^2 + 2^{2s}\left(\mathrm{Im}\left(\sum_{i=0}^{n} g_i\bar\alpha_i\right)\right)^2$$
$$= |g|^2 + 2^{2s}|g_{\bar\alpha}|^2.$$

This correspondence between polynomials in $\mathbf{Z}[X]$ of degree at most n and vectors in the lattice L_s is easily seen to be 1-1 onto and readily invertible. We will strongly separate the minimal polynomial $h(X)$ of α from all other polynomials $g(X)$ of degree n or less with $g(\alpha) \neq 0$ by showing that for a suitable choice of s and small enough $|\alpha^i - \bar\alpha_i|$,

$$|\tilde g|^2 > 2^n|\tilde h|^2.$$

We run the basis reduction algorithm on $b_0, b_1, \ldots, b_n$ to get a reduced basis. Suppose $\tilde v$ is the first vector of this basis, and $v(X)$ the corresponding polynomial. Because the degree of h was supposed to be equal to n, we have that $\tilde h$ is contained in L_s, so that $\Lambda_1(L_s) \le |\tilde h|$. Theorem (1.1) now yields $|\tilde v|^2 \le 2^n|\tilde h|^2$, and therefore $v(\alpha) = 0$ by the strong separation. This implies that h is a factor of v. Combining this with $\deg(v) \le \deg(h)$, we see that v and h are associates; further, the fact that $\tilde v$ belongs to a basis for L_s implies that $v = \pm h$.

The s needed will be bounded by a polynomial function of d and $\log H$. Here is a short intuitive description of how the strong separation is proved. If the powers of α are sufficiently close to the $\bar\alpha_i$, clearly $h_{\bar\alpha}$ is close to $h(\alpha) = 0$ (quantified in Lemma (1.5)). Thus $|\tilde h|^2 = |h|^2$+(a small term) and can be bounded above. To show that $|\tilde g|^2$ is large for other g, we consider two cases: If $|g|$ is large, then of course $|\tilde g|$ is large. If $|g|$ is small, then we show that $|g(\alpha)|$ has to be bounded from below (Proposition (1.6)). Again, $|g_{\bar\alpha}|$ being close to $|g(\alpha)|$, we are able to bound it from below and hence bound also $|\tilde g|^2$ from below.

(1.5) LEMMA. *If α and $\bar\alpha_i$ for $0 \le i \le n$ are complex numbers such that $\bar\alpha_0 = 1$, and $|\alpha^i - \bar\alpha_i| \le \varepsilon$ for $1 \le i \le n$ and f is a polynomial of degree at most n in $\mathbf{C}[X]$, then*

$$|f(\alpha) - f_{\bar\alpha}| \le \varepsilon\cdot n\cdot|f|_\infty.$$

Proof. Immediate.

(1.6) PROPOSITION. *Let h and g be nonzero polynomials in $\mathbf{Z}[X]$ of degrees n and m, respectively, and let $\alpha \in \mathbf{C}$ be a zero of h with $|\alpha| \leq 1$. If h is irreducible and $g(\alpha) \neq 0$ then*

$$|g(\alpha)| \geq n^{-1} \cdot |h|^{-m} \cdot |g|^{-n+1}.$$

Proof. Because h is nonzero and α is a zero of h we have that $n \geq 1$. If $m = 0$, then $g(\alpha) = |g|$, so that the result follows. Now assume that $m \neq 0$. Define the $(n+m) \times (n+m)$ matrix M as the matrix having ith column $X^{i-1} \cdot h$ for $1 \leq i \leq m$, and $X^{i-m-1} \cdot g$ for $m+1 \leq i \leq n+m$, where the polynomials $X^{i-1} \cdot h$ and $X^{i-m-1} \cdot g$ are regarded as $(n+m)$-dimensional vectors. By R we denote the absolute value of the determinant of M, the so-called *resultant* of h and g.

We prove that this resultant R is nonzero. Suppose on the contrary that the determinant of M is zero. This implies that a linear combination of the columns of M is zero, so that there exist polynomials $a, b \in \mathbf{Z}[X]$ with degree$(a) < m$ and degree$(b) < n$ such that $a \cdot h + b \cdot g = 0$. Because h is irreducible, any nontrivial common factor of h and g must have α as a zero, so that with $g(\alpha) \neq 0$ we have that $\gcd(h, g) = 1$. Therefore, we have that h divides b, so that with degree$(b) < n$, we find $b = 0$, and also $a = 0$. This proves that the columns of M are linearly independent, so that $R \neq 0$. Because the entries of M are integral, we even have $R \geq 1$.

We add, for $2 \leq i \leq n + m$, the ith row of M times T^{i-1} to the first row of M. The first row of M then becomes $(h(T), T \cdot h(T), \ldots, T^{m-1} \cdot h(T), g(T), T \cdot g(T), \ldots, T^{n-1} \cdot g(T))$. Expanding the determinant of M with respect to the first row, we find that

$$R = |h(T) \cdot (a_0 + a_1 \cdot T + \cdots + a_{m-1} \cdot T^{m-1}) + g(T) \cdot (b_0 + b_1 \cdot T + \cdots + b_{n-1} \cdot T^{n-1})|,$$

where the a_i and b_j are determinants of $(n + m - 1) \times (n + m - 1)$ submatrices of M. Evaluating the above identity for $T = \alpha$ yields

$$R = |g(\alpha)| \cdot |b_0 + b_1 \cdot \alpha + \cdots + b_{n-1} \cdot \alpha^{n-1}|, \tag{1.7}$$

because $h(\alpha) = 0$. From Hadamard's inequality it follows that $|b_j| \leq |h|^m \cdot |g|^{n-1}$. Combining this with $|\alpha| \leq 1$ we get

$$|b_0 + b_1 \cdot \alpha + \cdots + b_{n-1} \cdot \alpha^{n-1}| \leq n \cdot |h|^m \cdot |g|^{n-1},$$

so that (1.6) follows from (1.7) and $R \geq 1$:

$$|g(\alpha)| \geq n^{-1} \cdot |h|^{-m} \cdot |g|^{-n+1}.$$

This proves Proposition (1.6).

(1.8) *Remark.* Proposition (1.6) implies that two algebraic numbers that are not conjugates (conjugates are roots of the same irreducible polynomial in $\mathbf{Z}[X]$) cannot get very close. More precisely, suppose α and β satisfy distinct irreducible primitive polynomials $h(X)$ and $g(X)$, respectively, in $\mathbf{Z}[X]$, each of degree at most n. Without loss of generality suppose that $|\beta| < |\alpha| \leq 1$, and let $|\alpha - \beta|$ be γ. It is easy to see that $|g(\alpha) - g(\beta)| \leq \gamma \cdot |g|_\infty \cdot n(n-1)/2$. Now a lower bound on γ follows from Proposition (1.6). This kind of separation result also holds if α and β are conjugates (see for instance [21, Section 20]).

(1.9) LEMMA. *Suppose α is a complex number with $|\alpha| \leq 1$ and with minimal polynomial h of degree at most $d \geq 1$ and height at most H, and suppose $\bar{\alpha}_i$ satisfies $\bar{\alpha}_0 = 1$ and $|\alpha^i - \bar{\alpha}_i| \leq 2^{-s}$ for $1 \leq i \leq d$. Let g be a polynomial with integral coefficients of degree at most d such that $g(\alpha) \neq 0$. Then with the notation introduced in* (1.4), *the following inequalities hold*:

$$|\tilde{h}| \leq (d+1) \cdot H, \tag{1.10}$$

$$|\tilde{g}| > 2^{d/2} \cdot (d+1) \cdot H, \tag{1.11}$$

provided

$$2^s \geq 2^{d^2/2} \cdot (d+1)^{(3d+4)/2} \cdot H^{2d}. \tag{1.12}$$

Proof. First notice that

$$|f|^2 \leq (d+1) \cdot |f|_\infty^2 \tag{1.13}$$

holds for any polynomial f of degree at most d. To prove (1.10), we combine $|\tilde{h}|^2 = |h|^2 + 2^{2s}|h_{\bar{\alpha}}|^2$ and $|h_{\bar{\alpha}}| = |h(\alpha) - h_{\bar{\alpha}}| \leq 2^{-s} \cdot d \cdot H$ (Lemma (1.5)):

$$\begin{aligned} |\tilde{h}|^2 &\leq |h|^2 + d^2 \cdot H^2 \\ &\leq (d+1) \cdot H^2 + d^2 \cdot H^2 \quad \text{(cf. (1.13))} \\ &< (d+1)^2 \cdot H^2. \end{aligned}$$

This proves (1.10). We now prove (1.11). Clearly, if $|g| > 2^{d/2} \cdot (d+1) \cdot H$, we are done because $|\tilde{g}|^2 = |g|^2 + 2^{2s}|g_{\bar{\alpha}}|^2$. So assume $|g| \leq 2^{d/2} \cdot (d+1) \cdot H$. By Proposition (1.6) and (1.13),

$$\begin{aligned} g(\alpha) &\geq d^{-1} \cdot ((d+1) \cdot H^2)^{-d/2} \cdot (2^{d/2} \cdot (d+1) \cdot H)^{-d+1} \\ &> 2^{-d(d-1)/2} \cdot (d+1)^{-3d/2} \cdot H^{-2d+1}, \end{aligned}$$

so that, with Lemma (1.5) and $|\alpha^i - \bar{\alpha}_i| \leq 2^{-s}$:

$$\begin{aligned} |\tilde{g}| &\geq 2^s \cdot |g_{\bar{\alpha}}| \\ &\geq 2^s \cdot \left(2^{-d(d-1)/2} \cdot (d+1)^{-3d/2} \cdot H^{-2d+1} - 2^{-s} \cdot d \cdot |g|_\infty\right) \\ &= 2^s \cdot 2^{-d(d-1)/2} \cdot (d+1)^{-3d/2} \cdot H^{-2d+1} - d \cdot |g|_\infty. \end{aligned} \tag{1.14}$$

From (1.12) and $|2^{d/2} \cdot (d+1) \cdot H| \geq |g| \geq |g|_\infty$ we get

$$\begin{aligned} &2^s \cdot 2^{-d(d-1)/2} \cdot (d+1)^{-3d/2} \cdot H^{-2d+1} \\ &\quad \geq 2^{d/2} \cdot (d+1)^2 \cdot H = (d \cdot (d+1) + (d+1)) \cdot 2^{d/2} \cdot H \\ &\quad \geq d \cdot |g|_\infty + 2^{d/2} \cdot (d+1) \cdot H, \end{aligned}$$

which, combined with (1.14), yields (1.11). This proves Lemma (1.9).

(1.15) THEOREM. *Let $\alpha, h(X), d, H$, and $\bar{\alpha}_i \in 2^{-s}\mathbf{Z}\left[\sqrt{-1}\right]$, for $0 \leq i \leq d$, satisfy the hypothesis of Lemma* (1.9), *where s is such that* (1.12) *holds. Let n be an integer satisfying $1 \leq n \leq d$, and suppose that the basis reduction algorithm on input $b_0, b_1, \ldots, b_n$ defined in* (1.3) *yields a reduced basis with $\tilde{v} = \sum_{i=0}^n v_i b_i$ as the first vector. Then the following three assertions are equivalent*:

(i) $|\tilde{v}| \leq 2^{d/2} \cdot (d+1) \cdot H$;

(ii) *α satisfies the polynomial $v(X) = \sum_{i=0}^n v_i X^i$*;

(iii) *the degree of α is at most n.*

Furthermore, if n equals the degree of α, then $h(X) = \pm v(X)$.

Proof. First notice that the lattice $L_s = L(b_0, b_1, \ldots, b_n)$ is contained in $\mathbf{Z}^{n+3}$, so that Theorem (1.1) can be applied to L_s, and that the conditions for Lemma (1.9) are satisfied.

Assume (i). From Lemma (1.9) we get $v(\alpha) = 0$, which is (ii).

Next, assume (ii). Then α satisfies a polynomial of degree at most n, which is (iii).

Finally, assume (iii). This implies that h has degree at most n, so that $\tilde{h}$ is a well-defined vector in L_s. Lemma (1.9) yields $|\tilde{h}| \le (d+1) \cdot H$, so that in the notation of Theorem (1.1) we have $\Lambda_1(L_s) \le (d+1) \cdot H$. It then follows from Theorem (1.1) that $|\tilde{v}| \le 2^{d/2} \cdot (d+1) \cdot H$, which is (i). This proves the equivalence of (i), (ii), and (iii).

If n equals the degree of α, then (iii) is satisfied, so that α satisfies $v(X)$ (from (ii)). Because $\deg(h) = n, \deg(v) \le n$, and h is irreducible, we then have that v is an integral multiple of h. It follows that $h = \pm v$ because both $\tilde{h}$ and $\tilde{v}$ are contained in L_s, and because $\tilde{v}$ belongs to a basis for L_s. This proves Theorem (1.15).

This theorem leads to the following algorithm for finding the minimal polynomial of α:

(1.16) ALGORITHM MINIMAL POLYNOMIAL. Suppose we get on input upper bounds d and H on the degree and height, respectively, of an algebraic number α with $|\alpha| \le 1$ and a complex rational number $\bar{\alpha}$ approximating α such that $|\bar{\alpha}| \le 1$ and $|\alpha - \bar{\alpha}| \le 2^{-s}/(4d)$, where s is the smallest positive integer such that

$$2^s \ge 2^{d^2/2} \cdot (d+1)^{(3d+4)/2} \cdot H^{2d}.$$

First compute $\bar{\alpha}_i \in 2^{-s}\mathbf{Z}\left[\sqrt{-1}\right]$, for $0 \le i \le d$, such that $\bar{\alpha}_0 = 1$ and $|\bar{\alpha}^i - \bar{\alpha}_i| \le 2^{-s-1/2}$ for $1 \le i \le d$. This can be done by rounding the powers of $\bar{\alpha}$ to s bits after the binary point. (It is easily verified that the $\bar{\alpha}_i$ satisfy the conditions in Theorem (1.15), see Explanation (1.17).)

For $n = 1, 2, \ldots, d$ in succession we do the following:

- Apply the basis reduction algorithm to the lattice $L_s = L(b_0, b_1, \ldots, b_n)$ as defined in (1.3).
- If the first basis vector $\tilde{v}$ in the reduced basis satisfies $|\tilde{v}| \le 2^{d/2} \cdot (d+1) \cdot H$, then let $v(X)$ be the polynomial corresponding to $\tilde{v}$ by the relation defined in (1.4), return $v(X)$ as the minimal polynomial of α, and terminate the execution of Algorithm (1.16).

This finishes the description of Algorithm (1.16).

(1.17) *Explanation.* We show that the $\bar{\alpha}_i$ for $1 \leq i \leq d$ satisfy the conditions in Theorem (1.15), i.e., $|\alpha^i - \bar{\alpha}_i| \leq 2^{-s}$:

$$\begin{aligned}|\alpha^i - \bar{\alpha}_i| &\leq |\alpha^i - \bar{\alpha}^i| + |\bar{\alpha}^i - \bar{\alpha}_i| \\ &\leq |\alpha - \bar{\alpha}| \cdot \sum_{j=1}^{i} |\alpha|^{i-j} \cdot |\bar{\alpha}|^{j-1} + 2^{-s-1/2} \quad \text{(due to the rounding)} \\ &\leq \frac{d}{4d} \cdot 2^{-s} + 2^{-s-1/2} \\ &< 2^{-s}.\end{aligned}$$

(1.18) *Explanation.* It is no major restriction to consider α with $|\alpha| \leq 1$ only. Namely, if $\alpha \neq 0$ satisfies the polynomial $h(X) = \sum_{i=0}^{d} h_i X^i$, then $1/\alpha$ satisfies $\sum_{i=0}^{d} h_{d-i} X^i$. Furthermore, an ε-approximation $\bar{\alpha}$ to α with $|\alpha| > 1$ easily yields a $3 \cdot \varepsilon$-approximation $\bar{\beta}$ to $\beta = 1/\alpha$. Let $|\alpha - \bar{\alpha}| \leq \varepsilon$ with ε such that $0 < \varepsilon \leq 1/2$. Determine $\bar{\beta}$ such that $|\bar{\beta} - 1/\bar{\alpha}| < \varepsilon$; then

$$\begin{aligned}|\beta - \bar{\beta}| &\leq \left|\beta - \frac{1}{\bar{\alpha}}\right| + \left|\bar{\beta} - \frac{1}{\bar{\alpha}}\right| \leq \left|\frac{\alpha - \bar{\alpha}}{\alpha \cdot \bar{\alpha}}\right| + \varepsilon \\ &\leq \frac{\varepsilon}{|\alpha| \cdot |\bar{\alpha}|} + \varepsilon.\end{aligned}$$

Now $|\bar{\alpha}| \geq (1 - \varepsilon)|\alpha|$, so $|\bar{\alpha}| \geq |\alpha|/2 \geq 1/2$. So $|\beta - \bar{\beta}| \leq \varepsilon[2 + 1] = 3 \cdot \varepsilon$.

(1.19) THEOREM. *Let α be an algebraic number and let d and H be upper bounds on the degree and height, respectively, of α. Suppose that we are given an approximation $\bar{\alpha}$ to α such that $|\alpha - \bar{\alpha}| \leq 2^{-s}/(12d)$, where s is the smallest positive integer such that*

$$2^s \geq 2^{d^2/2} \cdot (d+1)^{(3d+4)/2} \cdot H^{2d}.$$

Then the minimal polynomial of α can be determined in $O(n_0 \cdot d^4 \cdot (d + \log H))$ arithmetic operations on integers having $O(d^2 \cdot (d + \log H))$ binary bits, where n_0 is the degree of α.

Proof. In order to be able to apply Algorithm (1.16), we replace α by $1/\alpha$ if necessary. It follows from Explanation (1.18) that $\bar{\alpha}$ then yields an approximation $\bar{\beta}$ to $\beta = 1/\alpha$ such that $|\beta - \bar{\beta}| \leq 2^{-s}/(4d)$.

Now apply Algorithm (1.16). For a particular value of n the logarithm of the length of the vectors b_i in the initial basis for the lattice $L_s = L(b_0, b_1, \ldots, b_n)$ is $O(d^2 + d \cdot \log H)$ due to the choice of s. Application of the basis reduction algorithm to L_s can therefore be done in $O(n \cdot d^4 \cdot (d + \log H))$ arithmetic operations on integers having $O(d^2 \cdot (d + \log H))$ binary bits.

When going from n to $n + 1$ in Algorithm (1.16), we do not have to restart the basis reduction algorithm for the new lattice: We just add a new vector b_{n+1} and a new dimension in which all the old vectors have a zero component, whereas b_{n+1} has component 1. It follows from this observation and [16, (1.37)] that the applications of the basis reduction algorithm for all $n \leq n_0$ together can be done in $O(n_0 \cdot d^4 \cdot (d + \log H))$ arithmetic operations on integers having $O(d^2 \cdot (d + \log H))$ binary bits.

This bound clearly also holds for the computation of the $\bar{\alpha}_i$, which proves Theorem (1.19).

(1.20) *Remark.* A. Schönhage [22] has shown that for the lattice and the basis in (1.3), the basis reduction algorithm only needs $O(n \cdot d^3 \cdot (d + \log H))$ arithmetic operations on integers having $O(d \cdot (d + \log H))$ binary bits. This implies that Algorithm (1.16) actually needs $O(n_0 \cdot d^3 \cdot (d + \log H))$ operations on $O(d \cdot (d + \log H))$-bit integers.

A further improvement of a factor d in the number of operations can be obtained by means of Schönhage's improved basis reduction algorithm [22]. The formulation of Algorithm (1.16) should however be modified slightly to incorporate this improvement, as the analogue of [16, (1.37)] does not hold for the improved basis reduction algorithm; for details we refer to [22]. For a more efficient algorithm for basis reduction see also a paper by Schnorr [20].

2. Ramifications. The algorithm of the preceding section can be interpreted as saying the following: Polynomially many bits of an algebraic number are sufficient to specify it completely (polynomially in the number of bits needed to write down its minimal polynomial). In a vague sense, then, the bits of algebraic numbers are not random, but are completely determined by the first polynomially many bits. We will not make this sense very precise here—the cryptography papers referred to below undertake this task, but we will attempt to provide an intuitive description of why the results of the previous section show that the bits of algebraic numbers are not '(secure) pseudorandom' bits in the terminology of cryptographers.

The question of when an (infinite) sequence of 'bits' (0's and 1's) is random has been raised for a long time, and various reasonable definitions have been provided. Since any such sequence may be considered to be the binary expansion of a real number between 0 and 1, a rewording of the question is: When are the bits of a real number random? (The phrase 'the bits of a real number' will mean the binary expansion of the fractional part of the number.) The classical definition was provided by Borel in 1909 [4]. The gist of it follows: Define a real number α to be *normal with respect to the base* 2 if for any natural number k, each of the 2^k possible 0-1 strings of length k occur with equal probability in the bits of α. A similar definition can be made for other bases. It was not difficult to show that most real numbers are normal. It was shown by Champernowne [7] in 1933 that the real number α_0 which equals the infinite decimal $.123456789101112\ldots$ (whose digits are obtained by juxtaposing the digits of the integers $1, 2, 3, 4, \ldots$) is normal to the base 10. Copeland and Erdős [6] generalized this to any basis and a class of reals including α_0 and $\alpha_1 = .2357111317\ldots$ whose digits are obtained by juxtaposing the digits of successive primes. An excellent discussion of the various classical definitions of when a sequence is random appears in [14, Section 3.5].

In several applications related to computer science one would like a notion of randomness that implies some kind of unpredictability. The importance of this for cryptography as well as complexity theory is discussed in [23], [3], and [25]. Some other relevant papers related to this discussion are [9] and [8]. Of course, the bits of the real number α_0 above are eminently predictable; thus intuitively, normalcy does not seem to be a good criterion for randomness in this setting. Besides this objection, there is another—we cannot really define randomness for one single real number and still have unpredictability. The model we have in mind is one where a player A presents a player B with some bits of a real number and B is trying to predict the next bit. If there is one fixed real, B can compute the bits as fast as A can, and all bits are clearly predictable. So we will have to consider a set of numbers. The simplest set is the set of rationals. Blum, Blum and Shub [2] have shown the following: If A announces that he is giving out the bits of a rational number with denominator at most H, then after seeing $2 \cdot \log_2 H$ bits of the rational number, B can figure out its fractional part and thus compute the other bits in polynomial time. Since A needed at least $\log_2 H$ bits to store the rational, he cannot get a pseudorandom sequence of length more than a constant (2) times the length of the 'seed'.

The main result of the preceding section may be restated as follows:

If A *announces that he is giving the bits of an algebraic number which is the root of an irreducible primitive polynomial of degree d or less with integral coefficients each of absolute value at most H, then after seeing $O(d^2 + d \cdot \log_2 H)$ bits,* B *can compute in deterministic polynomial time the polynomial and hence find for any n the nth bit of the algebraic number in time polynomial in the data and n (for the latter statement see also Section* 3).

Intuitively, our result can be interpreted as saying that the bits of algebraic numbers cannot form very long pseudorandom sequences, because after seeing a number of bits that is polynomial in the length of the seed (the seed in this case would be the polynomial held by A) the sequence can be easily and uniquely inferred. As mentioned earlier, the question of whether this can be done was first raised by M. Blum (private communication) who foresaw the importance of the notion of predictability.

Another ramification of the result of the preceding section is that computations involving algebraic numbers can be done in a natural way by representing algebraic numbers by suitable rational approximations. The traditional representation of algebraic numbers is by their minimal polynomials (see, for example, [24] or [17]). We now know an efficient method of converting the rational approximation representation to the minimal polynomial representation. (For the conversion in the other direction, see Section 3.) While it is not hard to see that computations in either representation can be changed to computations in the other without loss of efficiency (the running time will not change by more than a polynomial), the rational approximation method is closer to the intuitive notion of computation. For this reason we briefly sketch as an example a polynomial-time algorithm for finding a primitive element (see definitions below) of the rationals extended by two algebraics. Landau and Miller [15] gave in 1983 a polynomial-time algorithm for the same problem as part of their algorithm for testing solvability by radicals.

First we remark that if α and β are two algebraic numbers, then given sufficiently close approximations to both, we can find the minimal polynomial of β over $\mathbf{Q}(\alpha)$— the least-degree polynomial $p(X)$ with coefficients in $\mathbf{Q}(\alpha)$ satisfied by β. This is done as follows. Suppose the degree of α over $\mathbf{Q}$ is d; then clearly each coefficient of $p(X)$ can be taken to be a polynomial in α of degree at most $d-1$ with integral coefficients. Suppose the degree of β over $\mathbf{Q}(\alpha)$ is m (we try $m = 1, 2, \ldots$ in order). Then $p(X) = \sum_{i=0}^{m} \sum_{j=0}^{d-1} p_{ij} \alpha^j X^i$ for some $p_{ij} \in \mathbf{Z}$. We can turn the problem of finding the p_{ij} (i.e., the problem of finding the minimal integral dependence among the $\alpha^j \beta^i$ for $0 \leq j \leq d-1$ and $0 \leq i \leq m$) into a lattice problem in exactly the same way as we turned the problem of finding the minimal integral dependence among α^j for $0 \leq j \leq d$ into a lattice problem in the preceding section. In the interest of space we do not elaborate.

Suppose that α is algebraic over $\mathbf{Q}$ of degree d, and β is another algebraic number whose degree over $\mathbf{Q}(\alpha)$ is m, where d and m are determined as described above. The field $\mathbf{Q}(\alpha, \beta)$ obtained by adjoining α and β to the set of rationals is the set of all complex numbers expressible as polynomials in α and β with rational coefficients. It is known that this field has a primitive element γ, i.e., an element γ with the property that $\mathbf{Q}(\alpha, \beta) = \mathbf{Q}(\gamma)$, and indeed $\gamma = \alpha + l \cdot \beta$, where l is a nonnegative integer at most $d \cdot m$. It is also easy to see that if the degree of $\alpha + l \cdot \beta$ is $d \cdot m$ over $\mathbf{Q}$, then $\mathbf{Q}(\alpha + l \cdot \beta)$ must be equal to $\mathbf{Q}(\alpha, \beta)$. Thus we can use the algorithm of Section 1 to find the degree of $\alpha + l \cdot \beta$ over $\mathbf{Q}$ for $l = 0, 1, \ldots, d \cdot m$, given sufficiently close approximations to α and β, and thereby find the primitive element. It would be interesting to cast the entire algorithm for testing solvability by radicals into one that deals with explicit approximations to the algebraic numbers involved.

The idea of computing with algebraic numbers in this fashion needs to be explored further. While it is too early to say if the algorithms will be better in practice, they should yield good theoretical and/or empirical insights.

The method of finding the minimal polynomial of β over $\mathbf{Q}(\alpha)$ can be extended to finding algebraic dependence between any number of complex numbers. More exactly, let $\alpha_1, \alpha_2, \ldots, \alpha_t$ be (possibly transcendental) complex numbers given by sufficiently good approximations. Assume that we know an upper bound d on the degree and an upper bound H on the coefficients of a polynomial $f \in \mathbf{Z}[X_1, X_2, \ldots, X_t]$ with $f(\alpha_1, \alpha_2, \ldots, \alpha_t) = 0$. Then we can compute such a polynomial f in time polynomial in $\log H$ and $\binom{d+t-1}{d}$. (This latter number is polynomial in d for fixed t and in t for fixed d.) The precision to which the numbers α_i must be known is also a polynomial number of bits in $\log H$ and $\binom{d+t-1}{d}$.

This yields a factorization algorithm for multivariate polynomials: Given $f \in \mathbf{Z}[X_1, X_2, \ldots, X_t]$, substitute sufficiently large random numbers $s_2, s_3, \ldots, s_t$ for $X_2, X_3, \ldots, X_t$, compute an s_1 such that $f(s_1, s_2, \ldots, s_t) \approx 0$, and then find an algebraic dependence between $s_1, s_2, \ldots, s_t$. For $t = 2$, a slight variant of this idea is worked out in detail in [12].

Applications to Some Transcendental Numbers. The same technique can be applied to transcendental numbers of the form $\cos^{-1}(\alpha), \sin^{-1}(\alpha), \log(\alpha)$ etc., where α is an algebraic number. The number π is included in this class since it is the principal value (i.e., the value belonging to the interval $(0, \pi]$) of $\cos^{-1}(-1)$.

Suppose β is the principal value of $\cos^{-1}(\alpha)$ for some unknown α, which is, however, known to be algebraic of degree and height at most d and H, respectively. The question is: Can we infer (in deterministic polynomial time) the minimal polynomial of α from an approximation $\bar{\beta}$ to β? We show that if $|\beta - \bar{\beta}|$ is at most $\varepsilon = 2^{-s}/(24d)$, this can be done, where s is such that

$$2^s \geq 2^{d^2/2} \cdot (d+1)^{(3d+4)/2} \cdot H^{2d}$$

as usual. The argument is as follows. First we show that a good approximation to β gives us a good approximation to $\alpha = \cos(\beta)$:

$$|\cos(\beta) - \cos(\bar{\beta})| \leq \varepsilon \cdot \max\left\{ \left[\left| \frac{d}{dx} \cos(x) \right| \right]_{x=y} : y \text{ between } \beta \text{ and } \bar{\beta} \right\} \leq \varepsilon.$$

This can be utilized if we can compute $\cos(\bar{\beta})$ at least approximately. To do this, we employ the Taylor series expansion of the cosine function and the argument that the tail of the series is small, once we consider several terms of the series. For all y with $0 \leq y < 2\pi$ we have

$$\cos(y) = 1 - y^2/2! + y^4/4! - y^6/6! + y^8/8! - \dots,$$

and further

$$\begin{aligned} &|\cos(y) - (1 - y^2/2! + y^4/4! - \dots + y^{4k}/(4k)!)| \\ &\leq \frac{|y|^{4k+1}}{(4k+1)!} \cdot \max\left\{ \left[\left| \frac{d^{4k+1}}{dz^{4k+1}} \cos(z) \right| \right]_{z=x} : x \text{ between } 0 \text{ and } y \right\} \\ &\leq (2\pi)^{4k+1}/(4k+1)!. \end{aligned}$$

Let k equal the maximum of $\lceil -(\log \varepsilon)/4 \rceil$ and $\lceil \pi e^2/2 \rceil$. Then using Stirling's formula, we see that $(2\pi)^{4k+1}/(4k+1)! \leq \varepsilon$. Denoting

$$g(y) = 1 - y^2/2! + y^4/4! - \dots + y^{4k}/(4k)!,$$

we find that

$$|g(\bar{\beta}) - \cos(\beta)| \leq |g(\bar{\beta}) - \cos(\bar{\beta})| + |\cos(\bar{\beta}) - \cos(\beta)| \leq 2 \cdot \varepsilon.$$

Thus, in polynomial time we can compute from $\bar{\beta}$ an approximation $\bar{\alpha}$ to an unknown algebraic number α such that $|\alpha - \bar{\alpha}| \leq 2 \cdot \varepsilon = 2^{-s}/(12d)$, with s as above. Now Theorem (1.19) guarantees that we can find the minimal polynomial of α in polynomial time. This argument can be extended to the inverses of functions that satisfy the following two definitions.

(2.1) *Definition.* A complex-valued function f defined on a subset D of the complex numbers is *approximable* if there is a deterministic algorithm that, given a complex number x in D with rational real and imaginary parts and a natural number t, computes a complex number α satisfying $|\alpha - f(x)| \leq 2^{-t}$ in time bounded by a polynomial function of t and the number of bits of x.

(2.2) *Definition.* A complex-valued function f defined on a subset D of the complex numbers satisfies the *uniform Lipschitz condition* if there exist $\delta, M > 0$ such that $|f(x) - f(y)| \leq M \cdot |x - y|$ for any x, y in D with $|x - y| \leq \delta$.

(2.3) THEOREM. *Suppose a complex-valued function f defined on a subset D of the complex numbers is approximable and satisfies the uniform Lipschitz condition, for certain $\delta, M > 0$. There is an algorithm which, given a complex number $\bar{\beta}$ in D with rational real and imaginary parts and two natural numbers d and H, determines whether or not there is a complex number β in D satisfying*

(i) *$|\beta - \bar{\beta}| \leq \varepsilon$, with $\varepsilon = \min((24d \cdot 2^{d^2/2} M(d+1)^{(3d+4)/2} H^{2d})^{-1}, \delta)$, and*

(ii) *$f(\beta)$ is an algebraic number of degree at most d and height at most H. Further, if such a β exists, then $f(\beta)$ is unique, and the algorithm determines the* *minimal polynomial of $f(\beta)$. The algorithm works in time bounded by a polynomial function of $d, \log H$, and the number of bits of $\bar{\beta}$.*

Proof. First we show that if a β satisfying (i) and (ii) exists in D, then $f(\beta)$ is unique. Suppose not; then let β and γ satisfy (i) and (ii) and $f(\beta) \neq f(\gamma)$. Because $|\beta - \bar{\beta}| \leq \varepsilon$, we have that $|f(\beta) - f(\bar{\beta})| \leq \varepsilon \cdot M$ by the Lipschitz condition, and similarly $|f(\gamma) - f(\bar{\beta})| \leq \varepsilon \cdot M$. But then, $f(\beta)$ and $f(\gamma)$ are two algebraic numbers of degree at most d and height at most H with $|f(\beta) - f(\gamma)| \leq 2\varepsilon \cdot M$, contradicting the fact that distinct algebraic numbers cannot come too close (cf. Remark (1.8)). This proves the uniqueness of $f(\beta)$.

By the approximability of f we can compute $\bar{\alpha}$ such that $|\bar{\alpha} - f(\bar{\beta})| \leq \varepsilon \cdot M$. If a suitable β exists, then the Lipschitz condition gives $|f(\beta) - f(\bar{\beta})| \leq \varepsilon \cdot M$, so that $|f(\beta) - \bar{\alpha}| \leq 2\varepsilon \cdot M$. The proof now follows by Theorem (1.19).

The exponential function, sine function, hyperbolic sine and cosine functions, etc., when restricted to a finite interval (note that we need such a restriction for the exponential function), satisfy both definitions, and thus the theorem can be applied to them. At present, the only interesting consequence is the statement that the bits of reals of the form $\cos^{-1}(\alpha), \sin^{-1}(\alpha), \log(\alpha)$, where α is algebraic, do not form a pseudorandom sequence.

Notice that complex numbers of the form $\log(\alpha)$, where α is an algebraic number ($\neq 0, 1$), cannot be algebraic. This follows from the famous theorem of A. Baker [1] (on log linear forms).

3. Factorization of Polynomials.

In this section we describe an algorithm to factor primitive polynomials over the integers in polynomial time. The first polynomial-time algorithm for this was provided in [16]. As described in the introduction, our algorithm is conceptually simple—we find the roots of the given polynomial to a certain accuracy, and then find the minimal polynomials of the roots using the algorithm of Section 1. These must then be the irreducible factors of the given polynomial. Rabin [19, Section 3] first used such an idea to factor over finite fields, where it is possible to find the minimal polynomial of a root (which in general lies in an extension field) by solving a system of simultaneous linear equations. For polynomials with integral coefficients, an algorithm similar to ours is described in [5], without being polynomial-time, however.

Throughout this section, $f(X) \in \mathbf{Z}[X]$ is the given primitive polynomial to be factored, $\deg(f(X)) = d$. Let $H = \binom{d}{d/2} \cdot |f|$. In [18] it is shown that this H bounds the height of any factor in $\mathbf{Z}[X]$ of f (see also [14, Exercise 4.6.2.20]). The factoring algorithm now follows immediately from Algorithm (1.16).

(3.1) ALGORITHM FACTOR. Let f, d and H be as above. If $d \leq 1$, then return that f is irreducible and terminate the execution of the algorithm. Otherwise, do the following as long as $d \geq 2$:

- Let s be the smallest positive integer such that

$$2^s \geq 2^{d^2/2} \cdot (d+1)^{(3d+4)/2} \cdot H^{2d}$$

- Compute an approximation $\bar{\alpha}$ to a root α of f such that $|\alpha - \bar{\alpha}| \leq 2^{-s}/(12d)$ (this can be replaced by $2^{-s}/(4d)$ if $|\alpha| \leq 1$, cf. Explanation (1.18)), apply

Algorithm (1.16) to determine the minimal polynomial $h(X)$ of α, and return h as an irreducible factor of f.

– Replace d by $d - \deg(h)$, put $g(X) = f(X)/h(X)$ and return g as an irreducible factor of f if $d = 1$. Terminate the execution of the algorithm if $d \leq 1$; otherwise, replace f by g and go on.

This finishes the description of Algorithm (3.1).

It follows from Explanation (1.18), Theorem (1.19) and the definition of H that all application of Algorithm (1.16) together can be done in $O(d^5 \cdot (d + \log|f|))$ arithmetic operations on $O(d^2 \cdot (d + \log|f|))$-bit integers. A. Schönhage's observation (cf. Remark (1.20)) even brings this down to $O(d^4 \cdot (d + \log|f|))$ arithmetic operations on $O(d \cdot (d + \log|f|))$-bit integers.

It remains to analyze the cost of the computation of an approximation to a root of f. In [21] it is shown that the cost of computing approximations to all roots of f simultaneously, up to the precision needed in Algorithm (3.1), is dominated by the cost of the applications of Algorithm (1.16). This paper is however not yet published, and therefore we sketch how an approximation to a root of $f \in \mathbf{Z}[X]$ of degree d can be found in time polynomial in $d, \log|f|$ and the number of bits needed. The algorithm is due to A. Schönhage and is considerably slower than his method in [21]; we only include it to show that the problem can be solved in polynomial time. We need the following lemma, which follows from [10, Theorems 6.4b and 6.4e].

(3.2) LEMMA. *Let $g(X) = \sum_{i=0}^{d} g_i X^i \in \mathbf{C}[X]$, and let α be the root of g which is smallest in absolute value. If $R(g) = \min\{|g_0/g_m|^{1/m} : m \geq 1, g_m \neq 0\}$, then*

$$\frac{1}{2} \cdot R(g) \leq |\alpha| \leq d \cdot R(g).$$

Proof. If $g_0 = 0$, then $X = 0$ is a root, and the lemma is obviously true. So assume $g_0 \neq 0$. First, suppose that the lower bound on $|\alpha|$ is violated. Then

$$|\alpha| < \frac{1}{2}\left|\frac{g_0}{g_m}\right|^{1/m}$$

for all m with $g_m \neq 0$. So

$$\left|\sum_{m=1}^{d} g_m \alpha^m\right| \leq \sum_{m=1}^{d} |g_m||\alpha|^m < |g_0| \sum_{m=1}^{d} \frac{1}{2^m} < |g_0|.$$

This implies that we cannot have $\sum_{i=0}^{d} g_i \alpha^i = 0$, a contradiction.

Now suppose that $|\alpha| > d \cdot R(g)$. Let $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_d$ be the roots of g. Then

$$g_m = g_d \cdot \sum_{i_1, i_2, \ldots, i_{d-m}} \alpha_{i_1} \cdot \alpha_{i_2} \cdot \cdots \cdot \alpha_{i_{d-m}}$$

for $m = 0, 1, \ldots, d-1$, and in particular

$$g_0 = g_d \cdot \prod_{i=1}^{d} \alpha_i.$$

So,

$$\frac{g_m}{g_0} = \sum_{i_1, i_2, \ldots, i_m} \frac{1}{a_{i_1}} \cdot \frac{1}{\alpha_{i_2}} \cdot \cdots \cdot \frac{1}{\alpha_{i_m}}$$

for $m = 0, 1, \ldots, d-1, d$. Since $|\alpha_i| > d \cdot R(g)$ for $i = 1, 2, \ldots, d$, we have

$$\left|\frac{g_m}{g_0}\right| < \binom{d}{m} \left(\frac{1}{d \cdot R(g)}\right)^m \leq \frac{1}{R(g)^m}$$

for any m. It follows that

$$\left|\frac{g_0}{g_m}\right|^{1/m} > R(g)$$

for all m with $g_m \neq 0$. This is in contradiction with the definition of $R(g)$. This proves Lemma (3.2).

We now show how to approximate a root in polynomial time. We may assume that among the roots $\alpha_1, \alpha_2, \ldots, \alpha_d \in \mathbf{C}$ of f there is an α_i satisfying $|\alpha_i| \leq 1$ (otherwise, replace $f(X)$ by $X^d \cdot f(1/X)$). Let $t \in \mathbf{Z}_{\geq 0}$ and $a_t \in 2^{-t}\mathbf{Z}\left[\sqrt{-1}\right]$ such that

(3.3) $$\min_i |a_t - \alpha_i| \leq 4d \cdot 2^{-t}.$$

Initially, this condition is satisfied for $t = 0$ and $a_0 = 0$. We show how to compute $a_{t+1} \in 2^{-(t+1)}\mathbf{Z}\left[\sqrt{-1}\right]$ such that (3.3) holds with t replaced by $t+1$.

For all $a \in 2^{-(t+1)}\mathbf{Z}\left[\sqrt{-1}\right]$ such that

(3.4) $$|a - a_t| \leq 4d \cdot 2^{-t} + 2^{-(t+1)}$$

we compute the coefficients of $g_a(X) = f(X + a)$ and an approximation $r(g_a)$ to $d \cdot R(g_a)$ such that

(3.5) $$d \cdot R(g_a) \leq r(g_a) \leq 2d \cdot R(g_a),$$

where $R(g_a)$ is defined as in Lemma (3.2). Define a_{t+1} as the a for which $r(g_a)$ is minimal.

To prove that a_{t+1} satisfies (3.3) with t replaced by $t+1$, notice that the roots of $g_a(X)$ are the $\alpha_i - a$, and that it follows from (3.3) and (3.4) that there is an a' among the a such that $\min_i |a' - \alpha_i| \leq 2^{-(t+1)}$. This yields:

$$\begin{aligned}
\min_i |a_{t+1} - \alpha_i| &\leq r(g_{a_{t+1}}) \quad \text{(Lemma (3.2) and (3.5))} \\
&\leq r(g_a) \quad \text{(choice of } \alpha_{t+1}\text{)} \\
&\leq 2d \cdot R(g_a) \quad \text{(due to (3.5))} \\
&\leq 4d \cdot \min_i |a' - \alpha_i| \quad \text{(Lemma (3.2))} \\
&\leq 4d \cdot 2^{-(t+1)} \quad \text{(choice of } a'\text{)}.
\end{aligned}$$

It is clear that the computation of a_{t+1} can be done in time polynomial in d, t, and $\log|f|$. It follows that an approximation to a root of f can be found in time polynomial in $d, \log|f|$ and the number of bits needed.

We have shown the following theorem.

(3.6) THEOREM. *A primitive polynomial f of degree d in one variable with integral coefficients can be completely factored over the integers in time polynomial in d and* $\log|f|$.

Using A. Schönhage's observation mentioned in Remark (1.20) and his improved version of the polynomial-time root finding algorithm described above (cf. [21]), we get the following theorem.

(3.7) THEOREM. *A primitive polynomial f of degree d in one variable with integral coefficients can be completely factored over the integers in $O(d^4 \cdot (d+\log|f|))$ arithmetic operations on $O(d \cdot (d+\log|f|))$-bit integers.*

As mentioned in Remark (1.20), the number of operations can be reduced to $O(d^3 \cdot (d+\log|f|))$ if we use Schönhage's improved basis reduction algorithm. The description of the algorithm should in that case be slightly modified; we refer to [22] for details.

Acknowledgment. We gratefully acknowledge one referee for pointing out two errors in Section 2.

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Department of Computer Science
The University of Chicago
Chicago, Illinois 60637

Eőtvős Loránd University
Budapest, Hungary

1. A. BAKER, "Linear forms in the logarithms of algebraic numbers I, II, III, IV," *Mathematika*, v. 13, 1966, pp. 204–216; *ibid.*, v. 14, 1967, pp. 102–107; *ibid.*, v. 14, 1967, pp. 220–228; *ibid.*, v. 15, 1968, pp. 204–216.

2. L. BLUM, M. BLUM & M. SHUB, *A Simple Secure Pseudo Random Number Generator*, Proceedings of Crypto 82.

3. M. BLUM & S. MICALI, *How to Generate Cryptographically Strong Sequences of Pseudo Random Bits*, Proc. 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 112–117.

4. É. BOREL, *Leçons sur la Théorie des Fonctions*, 2nd ed., 1914, pp. 182–216.

5. A. J. BRENTJES, "Multi-dimensional continued fraction algorithms," in *Computational Methods in Number Theory* (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Math. Centre Tracts 154, 155, Mathematisch Centrum, Amsterdam, 1982.

6. A. H. COPELAND & P. ERDŐS, "Note on normal numbers," *Bull. Amer. Math. Soc.*, v. 52, 1946, pp. 857–860.

7. D. G. CHAMPERNOWNE, "The construction of decimals normal in the scale of ten," *J. London Math. Soc.*, v. 8, 1933, pp. 254–260.

8. O. GOLDREICH, S. GOLDWASSER & S. MICALI, *How to Construct Random Functions*, Proc. 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 464–479.

9. S. GOLDWASSER, S. MICALI & P. TONG, *Why and How to Establish a Private Code on a Public Network*, Proc. 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 134–144.

10. P. HENRICI, *Applied and Computational Complex Analysis*, vol. 1, Wiley, New York, 1974.

11. I. N. HERSTEIN, *Topics in Algebra*, 2nd ed., Xerox, 1976.

12. M.-P. VAN DER HULST & A. K. LENSTRA, *Polynomial Factorization by Transcendental Evaluation*, Proceedings Eurocal 85.

13. R. KANNAN, A. K. LENSTRA & L. LOVÁSZ, *Polynomial Factorization and Nonrandomness of Bits of Algebraic and Some Transcendental Numbers*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 191–200.

14. D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

15. S. LANDAU & G. MILLER, *Solvability by Radicals is in Polynomial Time*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 140–151.

16. A. K. LENSTRA, H. W. LENSTRA, JR. & L. LOVÁSZ, "Factoring polynomials with rational coefficients," *Math. Ann.*, v. 261, 1982, pp. 513–534.

17. R. LOOS, "Computing in algebraic extensions," *Computer Algebra* (B. Buchberger, G. Collins and R. Loos, eds.), Springer-Verlag, Berlin and New York, 1982, pp. 173–187.

18. M. MIGNOTTE, "An inequality about factors of polynomials," *Math. Comp.*, v. 28, 1974, pp. 1153–1157.

19. M. O. RABIN, "Probabilistic algorithms in finite fields," *SIAM J. Comput.*, v. 9, 1980, pp. 273–280.

20. C. P. SCHNORR, "A more efficient algorithm for lattice basis reduction," manuscript, 1985.

21. A. SCHÖNHAGE, *The Fundamental Theorem of Algebra in Terms of Computational Complexity*, Preliminary report, Math. Inst. Univ. Tübingen, 1982.

22. A. SCHÖNHAGE, *Factorization of Univariate Integer Polynomials by Diophantine Approximation and an Improved Basis Reduction Algorithm*, Proc. 11th International Colloquium on Automata, Languages, and Programming, 1984, LNCS 172, 1984, pp. 436–447.

23. A. SHAMIR, *On the Generation of Cryptographically Strong Pseudo-Random Sequences*, Proc. 8th International Colloquium on Automata, Languages, and Programming, 1981.

24. B. TRAGER, *Algebraic Factoring and Rational Function Integration*, Proc. SYMSAC 76, pp. 219–226.

25. A. YAO, *Theory and Applications of Trapdoor Functions*, Proc. 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 80–91.

Math. Ann. 261, 515–534 (1982)

Mathematische Annalen
© Springer-Verlag 1982

Factoring Polynomials with Rational Coefficients

A. K. Lenstra[1], H. W. Lenstra, Jr.[2], and L. Lovász[3]

1 Mathematisch Centrum, Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands
2 Mathematisch Instituut, Universiteit van Amsterdam, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands
3 Bolyai Institute, A. József University, Aradi vèrtanúk tere 1, H-6720 Szeged, Hungary

In this paper we present a polynomial-time algorithm to solve the following problem: given a non-zero polynomial $f \in \mathbb{Q}[X]$ in one variable with rational coefficients, find the decomposition of f into irreducible factors in $\mathbb{Q}[X]$. It is well known that this is equivalent to factoring *primitive* polynomials $f \in \mathbb{Z}[X]$ into irreducible factors in $\mathbb{Z}[X]$. Here we call $f \in \mathbb{Z}[X]$ primitive if the greatest common divisor of its coefficients (the *content* of f) is 1.

Our algorithm performs well in practice, cf. [8]. Its running time, measured in bit operations, is $O(n^{12} + n^9(\log|f|)^3)$. Here $f \in \mathbb{Z}[X]$ is the polynomial to be factored, $n = \deg(f)$ is the degree of f, and

$$\left|\sum_i a_i X^i\right| = \left(\sum_i a_i^2\right)^{1/2}$$

for a polynomial $\sum_i a_i X^i$ with real coefficients a_i.

An outline of the algorithm is as follows. First we find, for a suitable small prime number p, a p-adic irreducible factor h of f, to a certain precision. This is done with Berlekamp's algorithm for factoring polynomials over small finite fields, combined with Hensel's lemma. Next we look for the irreducible factor h_0 of f in $\mathbb{Z}[X]$ that is divisible by h. The condition that h_0 is divisible by h means that h_0 belongs to a certain lattice, and the condition that h_0 divides f implies that the coefficients of h_0 are relatively small. It follows that we must look for a "small" element in that lattice, and this is done by means of a basis reduction algorithm. It turns out that this enables us to determine h_0. The algorithm is repeated until all irreducible factors of f have been found.

The basis reduction algorithm that we employ is new, and it is described and analysed in Sect. 1. It improves the algorithm given in a preliminary version of [9, Sect. 3]. At the end of Sect. 1 we briefly mention two applications of the new algorithm to diophantine approximation.

The connection between factors of f and reduced bases of a lattice is treated in detail in Sect. 2. The theory presented here extends a result appearing in [8, Theorem 2]. It should be remarked that the latter result, which is simpler to prove, would in principle have sufficed for our purpose.

0025-5831/82/0261/0515/$04.00

Section 3, finally, contains the description and the analysis of our algorithm for factoring polynomials.

It may be expected that other irreducibility tests and factoring methods that depend on diophantine approximation (Cantor [3], Ferguson and Forcade [5], Brentjes [2, Sect. 4A], and Zassenhaus [16]) can also be made into polynomial-time algorithms with the help of the basis reduction algorithm presented in Sect. 1.

Splitting an arbitrary non-zero polynomial $f \in \mathbb{Z}[X]$ into its *content* and its *primitive part*, we deduce from our main result that the problem of factoring such a polynomial is polynomial-time reducible to the problem of factoring positive integers. The same fact was proved by Adleman and Odlyzko [1] under the assumption of several deep and unproved hypotheses from number theory.

The generalization of our result to algebraic number fields and to polynomials in several variables is the subject of future publications.

1. Reduced Bases for Lattices

Let n be a positive integer. A subset L of the n-dimensional real vector space $\mathbb{R}^n$ is called a *lattice* if there exists a basis $b_1, b_2, \ldots, b_n$ of $\mathbb{R}^n$ such that

$$L = \sum_{i=1}^{n} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{n} r_i b_i : r_i \in \mathbb{Z} (1 \leqq i \leqq n) \right\}.$$

In this situation we say that $b_1, b_2, \ldots, b_n$ form a *basis* for L, or that they *span* L. We call n the *rank* of L. The *determinant* $d(L)$ of L is defined by

$$d(L) = |\det(b_1, b_2, \ldots, b_n)|, \tag{1.1}$$

the b_i being written as column vectors. This is a positive real number that does not depend on the choice of the basis [4, Sect. 1.2].

Let $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$ be linearly independent. We recall the Gram-Schmidt orthogonalization process. The vectors b_i^* $(1 \leqq i \leqq n)$ and the real numbers μ_{ij} $(1 \leqq j < i \leqq n)$ are inductively defined by

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \tag{1.2}$$

$$\mu_{ij} = (b_i, b_j^*)/(b_j^*, b_j^*), \tag{1.3}$$

where $(\,,\,)$ denotes the ordinary inner product on $\mathbb{R}^n$. Notice that b_i^* is the projection of b_i on the orthogonal complement of $\sum_{j=1}^{i-1} \mathbb{R} b_j$, and that $\sum_{j=1}^{i-1} \mathbb{R} b_j = \sum_{j=1}^{i-1} \mathbb{R} b_j^*$, for $1 \leqq i \leqq n$. It follows that $b_1^*, b_2^*, \ldots, b_n^*$ is an orthogonal basis of $\mathbb{R}^n$.

In this paper, we call a basis $b_1, b_2, \ldots, b_n$ for a lattice L *reduced* if

$$|\mu_{ij}| \leqq 1/2 \quad \text{for} \quad 1 \leqq j < i \leqq n \tag{1.4}$$

and

$$|b_i^* + \mu_{i\,i-1} b_{i-1}^*|^2 \geqq \tfrac{3}{4} |b_{i-1}^*|^2 \quad \text{for} \quad 1 < i \leqq n, \tag{1.5}$$

where $|\,|$ denotes the ordinary Euclidean length. Notice that the vectors $b_i^* + \mu_{i\,i-1} b_{i-1}^*$ and b_{i-1}^* appearing in (1.5) are the projections of b_i and b_{i-1} on the orthogonal complement of $\sum_{j=1}^{i-2} \mathbb{R}b_j$. The constant $\frac{3}{4}$ in (1.5) is arbitrarily chosen, and may be replaced by any fixed real number y with $\frac{1}{4} < y < 1$.

(1.6) **Proposition.** *Let $b_1, b_2, \dots, b_n$ be a reduced basis for a lattice L in $\mathbb{R}^n$, and let $b_1^*, b_2^*, \dots, b_n^*$ be defined as above. Then we have*

$$|b_j|^2 \leqq 2^{i-1} \cdot |b_i^*|^2 \quad \textit{for} \quad 1 \leqq j \leqq i \leqq n, \tag{1.7}$$

$$d(L) \leqq \prod_{i=1}^{n} |b_i| \leqq 2^{n(n-1)/4} \cdot d(L), \tag{1.8}$$

$$|b_1| \leqq 2^{(n-1)/4} \cdot d(L)^{1/n}. \tag{1.9}$$

Remark. If $\frac{3}{4}$ in (1.5) is replaced by y, with $\frac{1}{4} < y < 1$, then the powers of 2 appearing in (1.7), (1.8) and (1.9) must be replaced by the same powers of $4/(4y-1)$.

Remark. From (1.8) we see that a reduced basis is also reduced in the sense of [9, (7)].

Proof of (1.6). From (1.5) and (1.4) we see that

$$|b_i^*|^2 \geqq (\tfrac{3}{4} - \mu_{i\,i-1}^2) \cdot |b_{i-1}^*|^2 \geqq \tfrac{1}{2} \cdot |b_{i-1}^*|^2$$

for $1 < i \leqq n$, so by induction

$$|b_j^*|^2 \leqq 2^{i-j} \cdot |b_i^*|^2 \quad \text{for} \quad 1 \leqq j \leqq i \leqq n.$$

From (1.2) and (1.4) we now obtain

$$\begin{aligned} |b_i|^2 &= |b_i^*|^2 + \sum_{j=1}^{i-1} \mu_{ij}^2 |b_j^*|^2 \\ &\leqq |b_i^*|^2 + \sum_{j=1}^{i-1} \tfrac{1}{4} 2^{i-j} |b_i^*|^2 \\ &= (1 + \tfrac{1}{4}(2^i - 2)) \cdot |b_i^*|^2 \\ &\leqq 2^{i-1} \cdot |b_i^*|^2. \end{aligned}$$

It follows that

$$|b_j|^2 \leqq 2^{j-1} \cdot |b_j^*|^2 \leqq 2^{i-1} \cdot |b_i^*|^2$$

for $1 \leqq j \leqq i \leqq n$. This proves (1.7).

From (1.1), (1.2) it follows that

$$d(L) = |\det(b_1^*, b_2^*, \dots, b_n^*)|$$

and therefore, since the b_i^* are pairwise orthogonal

$$d(L) = \prod_{i=1}^{n} |b_i^*|.$$

From $|b_i^*| \leqq |b_i|$ and $|b_i| \leqq 2^{(i-1)/2} \cdot |b_i^*|$ we now obtain (1.8). Putting $j = 1$ in (1.7) and taking the product over $i = 1, 2, \dots, n$ we find (1.9). This proves (1.6).

Remark. Notice that the proof of the inequality

$$(1.10) \qquad d(L) \leqq \prod_{i=1}^{n} |b_i|$$

did not require the basis to be reduced. This is *Hadamard's inequality.*

(1.11) **Proposition.** *Let $L \subset \mathbb{R}^n$ be a lattice with reduced basis $b_1, b_2, \ldots, b_n$. Then*

$$|b_1|^2 \leqq 2^{n-1} \cdot |x|^2$$

for every $x \in L$, $x \neq 0$.

Proof. Write $x = \sum_{i=1}^{n} r_i b_i = \sum_{i=1}^{n} r_i' b_i^*$ with $r_i \in \mathbb{Z}$, $r_i' \in \mathbb{R}$ $(1 \leqq i \leqq n)$. If i is the largest index with $r_i \neq 0$ then $r_i' = r_i$, so

$$|x|^2 \geqq r_i'^2 \cdot |b_i^*|^2 \geqq |b_i^*|^2.$$

By (1.7), we have $|b_1|^2 \leqq 2^{i-1} \cdot |b_i^*|^2 \leqq 2^{n-1} \cdot |b_i^*|^2$. This proves (1.11).

(1.12) **Proposition.** *Let $L \subset \mathbb{R}^n$ be a lattice with reduced basis $b_1, b_2, \ldots, b_n$. Let $x_1, x_2, \ldots, x_t \in L$ be linearly independent. Then we have*

$$|b_j|^2 \leqq 2^{n-1} \cdot \max\{|x_1|^2, |x_2|^2, \ldots, |x_t|^2\}$$

for $j = 1, 2, \ldots, t$.

Proof. Write $x_j = \sum_{i=1}^{n} r_{ij} b_i$ with $r_{ij} \in \mathbb{Z}$ $(1 \leqq i \leqq n)$ for $1 \leqq j \leqq t$. For fixed j, let $i(j)$ denote the largest i for which $r_{ij} \neq 0$. Then we have, by the proof of (1.11)

$$(1.13) \qquad |x_j|^2 \geqq |b_{i(j)}^*|^2$$

for $1 \leqq j \leqq t$. Renumber the x_j such that $i(1) \leqq i(2) \leqq \ldots \leqq i(t)$. We claim that $j \leqq i(j)$ for $1 \leqq j \leqq t$. If not, then $x_1, x_2, \ldots, x_j$ would all belong to $\mathbb{R}b_1 + Rb_2 + \ldots + \mathbb{R}b_{j-1}$, a contradiction with the linear independence of $x_1, x_2, \ldots, x_t$. From $j \leqq i(j)$ and (1.7) we obtain, using (1.13):

$$|b_j|^2 \leqq 2^{i(j)-1} \cdot |b_{i(j)}^*|^2 \leqq 2^{n-1} \cdot |b_{i(j)}^*|^2 \leqq 2^{n-1} \cdot |x_j|^2$$

for $j = 1, 2, \ldots, t$. This proves (1.12).

Remark. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ denote the successive minima of $|\,|^2$ on L, see [4, Chap. VIII], and let $b_1, b_2, \ldots, b_n$ be a reduced basis for L. Then (1.7) and (1.12) easily imply that

$$2^{1-i}\lambda_i \leqq |b_i|^2 \leqq 2^{n-1}\lambda_i \quad \text{for} \quad 1 \leqq i \leqq n,$$

so $|b_i|^2$ is a reasonable approximation of λ_i.

(1.14) *Remark.* Notice that the number 2^{n-1} may in (1.11) be replaced by $\max\{|b_1|^2/|b_i^*|^2 : 1 \leqq i \leqq n\}$ and in (1.12) by $\max\{|b_j|^2/|b_i^*|^2 : 1 \leqq j \leqq i \leqq n\}$.

(1.15) We shall now describe an algorithm that transforms a given basis $b_1, b_2, \ldots, b_n$ for a lattice L into a reduced one. The algorithm improves the algorithm given in a preliminary version of [9, Sect. 3]. Our description incorporates an additional improvement due to J. J. M. Cuppen, reducing our running time estimates by a factor n.

To initialize the algorithm we compute b_i^* $(1 \leqq i \leqq n)$ and μ_{ij} $(1 \leqq j < i \leqq n)$ using (1.2) and (1.3). In the course of the algorithm the vectors $b_1, b_2, \ldots, b_n$ will be changed several times, but always in such a way that they form a basis for L. After every change of the b_i we shall update the b_i^* and μ_{ij} in such a way that (1.2) and (1.3) remain valid.

At each step of the algorithm we shall have a current subscript $k \in \{1, 2, \ldots, n+1\}$. We begin with $k=2$.

We shall now iterate a sequence of steps that starts from, and returns to, a situation in which the following conditions are satisfied:

(1.16) $$|\mu_{ij}| \leqq \tfrac{1}{2} \quad \text{for} \quad 1 \leqq j < i < k,$$

(1.17) $$|b_i^* + \mu_{i\,i-1} b_{i-1}^*|^2 \geqq \tfrac{3}{4} |b_{i-1}^*|^2 \quad \text{for} \quad 1 < i < k.$$

These conditions are trivially satisfied if $k=2$.

In the above situation one proceeds as follows. If $k=n+1$ then the basis is reduced, and the algorithm terminates. Suppose now that $k \leqq n$. Then we first achieve that

(1.18) $$|\mu_{k\,k-1}| \leqq \tfrac{1}{2} \quad \text{if} \quad k > 1.$$

If this does not hold, let r be the integer nearest to $\mu_{k\,k-1}$, and replace b_k by $b_k - r b_{k-1}$. The numbers μ_{kj} with $j < k-1$ are then replaced by $\mu_{kj} - r\mu_{k-1\,j}$, and $\mu_{k\,k-1}$ by $\mu_{k\,k-1} - r$. The other μ_{ij} and all b_i^* are unchanged. After this change (1.18) holds.

Next we distinguish two cases.

Case 1. Suppose that $k \geqq 2$ and

(1.19) $$|b_k^* + \mu_{k\,k-1} b_{k-1}^*|^2 < \tfrac{3}{4} |b_{k-1}^*|^2.$$

Then we interchange b_{k-1} and b_k, and we leave the other b_i unchanged. The vectors b_{k-1}^* and b_k^* and the numbers $\mu_{k\,k-1}$, $\mu_{k-1\,j}$, μ_{kj}, μ_{ik-1}, μ_{ik}, for $j < k-1$ and for $i > k$, have now to be replaced. This is done by formulae that we give below. The most important one of these changes is that b_{k-1}^* is replaced by $b_k^* + \mu_{k\,k-1} b_{k-1}^*$; so the new value of $|b_{k-1}^*|^2$ is less than $\frac{3}{4}$ times the old one. These changes being made, we replace k by $k-1$. Then we are in the situation described by (1.16) and (1.17), and we proceed with the algorithm from there.

Case 2. Suppose that $k=1$ or

(1.20) $$|b_k^* + \mu_{k\,k-1} b_{k-1}^*|^2 \geqq \tfrac{3}{4} |b_{k-1}^*|^2.$$

In this case we first achieve that

(1.21) $$|\mu_{kj}| \leqq \tfrac{1}{2} \quad \text{for} \quad 1 \leqq j \leqq k-1.$$

[For $j=k-1$ this is already true, by (1.18).] If (1.21) does not hold, let l be the largest index $<k$ with $|\mu_{kl}| > \frac{1}{2}$, let r be the integer nearest to μ_{kl}, and replace b_k by

$b_k - rb_l$. The numbers μ_{kj} with $j<l$ are then replaced by $\mu_{kj} - r\mu_{lj}$, and μ_{kl} by $\mu_{kl} - r$; the other μ_{ij} and all b_i^* are unchanged. This is repeated until (1.21) holds.

Next we replace k by $k+1$. Then we are in the situation described by (1.16) and (1.17), and we proceed with the algorithm from there.

Notice that in the case $k=1$ we have done no more than replacing k by 2.

This finishes the description of the algorithm. Below we shall prove that the algorithm terminates.

(1.22) For the sake of completeness we now give the formulae that are needed in case 1. Let $b_1, b_2, \ldots, b_n$ be the current basis and b_i^*, μ_{ij} as in (1.2) and (1.3). Let k be the current subscript for which (1.16), (1.17), (1.18), and (1.19) hold. By c_i, c_i^*, and v_{ij} we denote the vectors and numbers that will replace b_i, b_i^*, and μ_{ij}, respectively. The new basis $c_1, c_2, \ldots, c_n$ is given by

$$c_{k-1} = b_k, \quad c_k = b_{k-1}, \quad c_i = b_i \quad \text{for} \quad i \neq k-1, k.$$

Since c_{k-1}^* is the projection of b_k on the orthogonal complement of $\sum_{j=1}^{k-2} \mathbb{R} b_j$ we have, as announced:

$$c_{k-1}^* = b_k^* + \mu_{k\,k-1} b_{k-1}^*$$

[cf. the remark after (1.5)]. To obtain c_k^* we must project b_{k-1}^* on the orthogonal complement of $\mathbb{R}c_{k-1}^*$. That leads to

$$\begin{aligned} v_{k\,k-1} &= (b_{k-1}^*, c_{k-1}^*)/(c_{k-1}^*, c_{k-1}^*) \\ &= \mu_{k\,k-1} |b_{k-1}^*|^2 / |c_{k-1}^*|^2, \\ c_k^* &= b_{k-1}^* - v_{k\,k-1} c_{k-1}^*. \end{aligned}$$

For $i \neq k-1$, k we have $c_i^* = b_i^*$. Let now $i > k$. To find $v_{i\,k-1}$ and v_{ik} we substitute

$$\begin{aligned} b_{k-1}^* &= v_{k\,k-1} c_{k-1}^* + c_k^* \\ b_k^* &= (1 - \mu_{k\,k-1} v_{k\,k-1}) c_{k-1}^* - \mu_{k\,k-1} c_k^* \\ &= (|b_k^*|^2 / |c_{k-1}^*|^2) \cdot c_{k-1}^* - \mu_{k\,k-1} c_k^* \end{aligned}$$

in $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^*$. That yields

$$\begin{aligned} v_{i\,k-1} &= \mu_{i\,k-1} v_{k\,k-1} + \mu_{ik} |b_k^*|^2 / |c_{k-1}^*|^2 \\ v_{ik} &= \mu_{i\,k-1} - \mu_{ik} \mu_{k\,k-1}. \end{aligned}$$

Finally, we have

$$v_{k-1\,j} = \mu_{kj}, \quad v_{kj} = \mu_{k-1\,j}$$

for $1 \leq j < k-1$, and $v_{ij} = \mu_{ij}$ if $1 \leq j < i \leq n$, $\{i, j\} \cap \{k-1, k\} = \emptyset$.

We remark that after the initialization stage of the algorithm it is not necessary to keep track of the vectors b_i^*. It suffices to keep track of the numbers $|b_i^*|^2$, in addition to μ_{ij} and the vectors b_i. Notice that $|c_k^*|^2 = |b_{k-1}^*|^2 \cdot |b_k^*|^2 / |c_{k-1}^*|^2$ in the above, and that the left hand side of (1.19), (1.20) equals $|b_k^*|^2 + \mu_{k\,k-1}^2 |b_{k-1}^*|^2$.

The entire algorithm is represented in Fig. 1, in which $B_i = |b_i^*|^2$.

```
    $\left.\begin{array}{l} b_i^* := b_i; \\ \left.\begin{array}{l}\mu_{ij} := (b_i, b_j^*)/B_j; \\ b_i^* := b_i^* - \mu_{ij} b_j^*\end{array}\right\}$ for $j = 1, 2, \ldots, i-1;$ \\ B_i := (b_i^*, b_i^*) \end{array}\right\}$ for $i = 1, 2, \ldots, n;$
    $k := 2;$
(1) perform (*) for $l = k-1;$
    if $B_k < (\frac{3}{4} - \mu_{k\,k-1}^2) B_{k-1}$, go to (2);
    perform (*) for $l = k-2, k-3, \ldots, 1;$
    if $k = n$, terminate;
    $k := k+1;$
    go to (1);
(2) $\mu := \mu_{k\,k-1}$; $B := B_k + \mu^2 B_{k-1}$; $\mu_{k\,k-1} := \mu B_{k-1}/B$;
    $B_k := B_{k-1} B_k / B$; $B_{k-1} := B$;
    $\begin{pmatrix} b_{k-1} \\ b_k \end{pmatrix} := \begin{pmatrix} b_k \\ b_{k-1} \end{pmatrix};$
    $\begin{pmatrix} \mu_{k-1\,j} \\ \mu_{kj} \end{pmatrix} := \begin{pmatrix} \mu_{kj} \\ \mu_{k-1\,j} \end{pmatrix}$ for $j = 1, 2, \ldots, k-2;$
    $\begin{pmatrix} \mu_{i\,k-1} \\ \mu_{ik} \end{pmatrix} := \begin{pmatrix} 1 & \mu_{k\,k-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} \mu_{i\,k-1} \\ \mu_{ik} \end{pmatrix}$ for $i = k+1, k+2, \ldots, n;$
    if $k > 2$, then $k := k-1;$
    go to (1).
(*) If $|\mu_{kl}| > \frac{1}{2}$, then:
    $\left\{\begin{array}{l} r := \text{integer nearest to } \mu_{kl};\ b_k := b_k - r b_l; \\ \mu_{kj} := \mu_{kj} - r\mu_{lj} \text{ for } j = 1, 2, \ldots, l-1; \\ \mu_{kl} := \mu_{kl} - r. \end{array}\right.$
```

Fig. 1. The reduction algorithm

(1.23) To prove that the algorithm terminates we introduce the quantities

(1.24) $$d_i = \det((b_j, b_l))_{1 \leq j, l \leq i}$$

for $0 \leqq i \leqq n$. It is easily checked that

(1.25) $$d_i = \prod_{j=1}^{i} |b_j^*|^2$$

for $0 \leqq i \leqq n$. Hence the d_i are positive real numbers. Notice that $d_0 = 1$ and $d_n = d(L)^2$. Put

$$D = \prod_{i=1}^{n-1} d_i.$$

By (1.25), the number D only changes if some b_i^* is changed, which only occurs in case 1. In case 1, the number d_{k-1} is reduced by a factor $< \frac{3}{4}$, by (1.25), whereas the other d_i are unchanged, by (1.24); hence D is reduced by a factor $< \frac{3}{4}$. Below we prove that there is a positive lower bound for d_i that only depends on L. It follows that there is also a positive lower bound for D, and hence an upper bound for the number of times that we pass through case 1.

In case 1, the value of k is decreased by 1, and in case 2 it is increased by 1. Initially we have $k=2$, and $k \leqq n+1$ throughout the algorithm. Therefore the number of times that we pass through case 2 is at most $n-1$ more than the number of times that we pass through case 1, and consequently it is bounded. This implies that the algorithm terminates.

To prove that d_i has a lower bound we put

$$m(L) = \min\{|x|^2 : x \in L, x \neq 0\}.$$

This is a positive real number. For $i>0$, we can interpret d_i as the square of the determinant of the lattice of rank i spanned by $b_1, b_2, \ldots, b_i$ in the vector space $\sum_{j=1}^{i} \mathbb{R} b_j$. By [4, Chap. I, Lemma 4 and Chap. II, Theorem I], this lattice contains a non-zero vector x with $|x|^2 \leqq (4/3)^{(i-1)/2} d_i^{1/i}$. Therefore $d_i \geqq (3/4)^{i(i-1)/2} m(L)^i$, as required.

We shall now analyse the running time of the algorithm under the added hypothesis that $b_i \in \mathbb{Z}^n$ for $1 \leqq i \leqq n$. By an *arithmetic operation* we mean an addition, subtraction, multiplication or division of two integers. Let the *binary length* of an integer a be the number of binary digits of $|a|$.

(1.26) **Proposition.** *Let $L \subset \mathbb{Z}^n$ be a lattice with basis $b_1, b_2, \ldots, b_n$, and let $B \in \mathbb{R}$, $B \geqq 2$, be such that $|b_i|^2 \leqq B$ for $1 \leqq i \leqq n$. Then the number of arithmetic operations needed by the basis reduction algorithm described in* (1.15) *is $O(n^4 \log B)$, and the integers on which these operations are performed each have binary length $O(n \log B)$.*

Remark. Using the classical algorithms for the arithmetic operations we find that the number of bit operations needed by the basis reduction algorithm is $O(n^6 (\log B)^3)$. This can be reduced to $O(n^{5+\varepsilon} (\log B)^{2+\varepsilon})$, for every $\varepsilon > 0$, if we employ fast multiplication techniques.

Proof of (1.26). We first estimate the number of times that we pass through cases 1 and 2. In the beginning of the algorithm we have $d_i \leqq B^i$, by (1.25), so $D \leqq B^{n(n-1)/2}$. Throughout the algorithm we have $D \geqq 1$, since $d_i \in \mathbb{Z}$ by (1.24) and $d_i > 0$ by (1.25). So by the argument in (1.23) the number of times that we pass through case 1 is $O(n^2 \log B)$, and the same applies to case 2.

The initialization of the algorithm takes $O(n^3)$ arithmetic operations with rational numbers; below we shall see how they can be replaced by operations with integers.

For (1.18) we need $O(n)$ arithmetic operations, and this is also true for case 1. In case 2 we have to deal with $O(n)$ values of l, that each require $O(n)$ arithmetic operations. Since we pass through these cases $O(n^2 \log B)$ times we arrive at a total of $O(n^4 \log B)$ arithmetic operations.

In order to represent all numbers that appear in the course of the algorithm by means of *integers* we also keep track of the numbers d_i defined by (1.24). In the initialization stage these can be calculated by (1.25). After that, they are only changed in case 1. In that case, d_{k-1} is replaced by $d_{k-1} \cdot |c_{k-1}^*|^2 / |b_{k-1}^*|^2 = d_{k-2} \cdot |c_{k-1}^*|^2$ [in the notation of (1.22)] whereas the other d_i are unchanged. By (1.24), the d_i are *integers*, and we shall now see that they can be used as denominators for all numbers that appear:

(1.27) $$|b_i^*|^2 = d_i/d_{i-1} \quad (1 \leqq i \leqq n),$$

(1.28) $$d_{i-1}b_i^* \in L \subset \mathbb{Z}^n \quad (1 \leqq i \leqq n),$$

(1.29) $$d_j\mu_{ij} \in \mathbb{Z} \quad (1 \leqq j < i \leqq n).$$

The first of these follows from (1.25). For the second, we write $b_i^* = b_i - \sum_{j=1}^{i-1} \lambda_{ij}b_j$ with $\lambda_{ij} \in \mathbb{R}$. Solving $\lambda_{i1}, \ldots, \lambda_{i\,i-1}$ from the system

$$(b_i, b_l) = \sum_{j=1}^{i-1} \lambda_{ij}(b_j, b_l) \quad (1 \leqq l \leqq i-1)$$

and using (1.24) we find that $d_{i-1}\lambda_{ij} \in \mathbb{Z}$, whence (1.28). Notice that the same argument yields

$$d_{i-1}\left(b_k - \sum_{j=1}^{i-1} \mu_{kj}b_j^*\right) \in \mathbb{Z}^n \quad \text{for} \quad i \leqq k;$$

this is useful for the calculation of b_k^* at the beginning of the algorithm. To prove (1.29) we use (1.3), (1.27), and (1.28):

$$d_j\mu_{ij} = d_j(b_i, b_j^*)/(b_j^*, b_j^*) = d_{j-1}(b_i, b_j^*) = (b_i, d_{j-1}b_j^*) \in \mathbb{Z}.$$

To finish the proof of (1.26) we estimate all integers that appear. Since no d_i is ever increased we have $d_i \leqq B^i$ throughout the algorithm. This estimates the denominators. To estimate the numerators it suffices to find upper bounds for $|b_i^*|^2$, $|b_i|^2$, and $|\mu_{ij}|$.

At the beginning we have $|b_i^*|^2 \leqq |b_i|^2 \leqq B$, and $\max\{|b_i^*|^2 : 1 \leqq i \leqq n\}$ is non-increasing; to see this, use that $|c_{k-1}^*|^2 < \frac{3}{4}|b_{k-1}^*|^2$ and $|c_k^*|^2 \leqq |b_{k-1}^*|^2$ in (1.22), the latter inequality because c_k^* is a projection of b_{k-1}^*. Hence we have $|b_i^*|^2 \leqq B$ throughout the algorithm.

To deal with $|b_i|^2$ and μ_{ij} we first prove that every time we arrive at the situation described by (1.16) and (1.17) the following inequalities are satisfied:

(1.30) $|b_i|^2 \leqq nB$ for $i \neq k$,

(1.31) $|b_k|^2 \leqq n^2(4B)^n$ if $k \neq n+1$,

(1.32) $|\mu_{ij}| \leqq \frac{1}{2}$ for $1 \leqq j < i,\ i < k$,

(1.33) $|\mu_{ij}| \leqq (nB^j)^{1/2}$ for $1 \leqq j < i,\ i > k$,

(1.34) $|\mu_{kj}| \leqq 2^{n-k}(nB^{n-1})^{1/2}$ for $1 \leqq j < k$, if $k \neq n+1$.

Here (1.30), for $i < k$, is trivial from (1.32), and (1.31) follows from (1.34). Using that

(1.35) $$\mu_{ij}^2 \leqq |b_i|^2/|b_j^*|^2 = d_{j-1}|b_i|^2/d_j \leqq B^{j-1}|b_i|^2$$

we see that (1.33) follows from (1.30), and (1.32) is the same as (1.16). It remains to prove (1.30) for $i > k$ and to prove (1.34). At the beginning of the algorithm we even have $|b_i|^2 \leqq B$ and $\mu_{ij}^2 \leqq B^j$, by (1.35), so it suffices to consider the situation at the end of cases 1 and 2. Taking into account that k changes in these cases, we see that in case 1 the set of vectors $\{b_i : i \neq k\}$ is unchanged, and that in case 2 the set $\{b_i : i > k\}$ is replaced by a subset. Hence the inequalities (1.30) are preserved. At the end of case 2, the new values for μ_{kj} (if $k \neq n+1$) are the old values of $\mu_{k+1\,j}$, so here (1.34) follows from the inequality (1.33) at the previous stage. To prove (1.34) at the end of case 1 we assume that it is valid at the previous stage, and we follow what happens to μ_{kj}. To achieve (1.18) it is, for $j < k-1$, replaced by $\mu_{kj} - r\mu_{k-1\,j}$, with $|r| < 2|\mu_{k\,k-1}|$ and $|\mu_{k-1\,j}| \leqq \frac{1}{2}$, so

$$(1.36) \qquad \begin{aligned} |\mu_{kj} - r\mu_{k-1\,j}| &\leqq |\mu_{kj}| + |\mu_{k\,k-1}| \\ &\leqq 2^{n-k+1}(nB^{n-1})^{1/2} \quad \text{by (1.34)}. \end{aligned}$$

In the notation of (1.22) we therefore have

$$|v_{k-1\,j}| \leqq 2^{n-(k-1)}(nB^{n-1})^{1/2} \quad \text{for} \quad j < k-1$$

and since $k-1$ is the new value for k this is exactly the inequality (1.34) to be proved.

Finally, we have to estimate $|b_i|^2$ and μ_{ij} at the other points in the algorithm. For this it suffices to remark that the maximum of $|\mu_{k1}|, |\mu_{k2}|, \ldots, |\mu_{k\,k-1}|$ is at most doubled when (1.18) is achieved, by (1.36), and that the same thing happens in case 2 for at most $k-2$ values of l. Combining this with (1.34) and (1.33) we conclude that throughout the course of the algorithm we have

$$|\mu_{ij}| \leqq 2^{n-1}(nB^{n-1})^{1/2} \quad \text{for} \quad 1 \leqq j < i \leqq n$$

and therefore

$$|b_i|^2 \leqq n^2(4B)^n \quad \text{for} \quad 1 \leqq i \leqq n.$$

This finishes the proof of (1.26).

(1.37) *Remark.* Let $1 \leqq n' \leqq n$. If k, in the situation described by (1.16) and (1.17), is for the first time equal to $n'+1$, then the first n' vectors $b_1, b_2, \ldots, b_{n'}$ form a reduced basis for the lattice of rank n' spanned by the first n' vectors of the initially given basis. This will be useful in Sect. 3.

(1.38) *Remark.* It is easily verified that, apart from some minor changes, the analysis of our algorithm remains valid if the condition $L \subset \mathbb{Z}^n$ is replaced by the condition that $(x, y) \in \mathbb{Z}$ for all $x, y \in L$; or, equivalently, that $(b_i, b_j) \in \mathbb{Z}$ for $1 \leqq i, j \leqq n$. The weaker condition that $(b_i, b_j) \in \mathbb{Q}$, for $1 \leqq i, j \leqq n$, is also sufficient, but in this case we should clear denominators before applying (1.26).

We close this section with two applications of our reduction algorithm. The first is to simultaneous diophantine approximation. Let n be a positive integer, $\alpha_1, \alpha_2, \ldots, \alpha_n$ real numbers, and $\varepsilon \in \mathbb{R}$, $0 < \varepsilon < 1$. It is a classical theorem [4, Sect.V.10] that there exist integers $p_1, p_2, \ldots, p_n, q$ satisfying

$$|p_i - q\alpha_i| \leqq \varepsilon \quad \text{for} \quad 1 \leqq i \leqq n,$$
$$1 \leqq q \leqq \varepsilon^{-n}.$$

We show that there exists a polynomial-time algorithm to find integers that satisfy a slightly weaker condition.

(1.39) **Proposition.** *There exists a polynomial-time algorithm that, given a positive integer n and rational numbers $\alpha_1, \alpha_2, \ldots, \alpha_n, \varepsilon$ satisfying $0<\varepsilon<1$, finds integers $p_1, p_2, \ldots, p_n, q$ for which*

$$|p_i - q\alpha_i| \leqq \varepsilon \quad \textit{for} \quad 1 \leqq i \leqq n,$$
$$1 \leqq q \leqq 2^{n(n+1)/4}\varepsilon^{-n}.$$

Proof. Let L be the lattice of rank $n+1$ spanned by the columns of the $(n+1)\times(n+1)$-matrix

$$\begin{pmatrix} 1 & 0 & \ldots & 0 & -\alpha_1 \\ 0 & 1 & \ldots & 0 & -\alpha_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -\alpha_n \\ 0 & 0 & \ldots & 0 & 2^{-n(n+1)/4}\varepsilon^{n+1} \end{pmatrix}.$$

The inner product of any two columns is rational, so by (1.38) there is a polynomial-time algorithm to find a reduced basis $b_1, b_2, \ldots, b_{n+1}$ for L. By (1.9) we then have

$$|b_1| \leqq 2^{n/4}\cdot d(L)^{1/(n+1)} = \varepsilon.$$

Since $b_1 \in L$, we can write

$$b_1 = (p_1 - q\alpha_1, p_2 - q\alpha_2, \ldots, p_n - q\alpha_n, q\cdot 2^{-n(n+1)/4}\varepsilon^{n+1})^{\mathsf{T}}$$

with $p_1, p_2, \ldots, p_n, q \in \mathbb{Z}$. It follows that

$$|p_i - q\alpha_i| \leqq \varepsilon \quad \text{for} \quad 1 \leqq i \leqq n,$$
$$|q| \leqq 2^{n(n+1)/4}\varepsilon^{-n}.$$

From $\varepsilon<1$ and $b_1 \neq 0$ we see that $q \neq 0$. Replacing b_1 by $-b_1$, if necessary, we can achieve that $q>0$.

This proves (1.39).

Another application of our reduction algorithm is to the problem of finding $\mathbb{Q}$-linear relations among given real numbers $\alpha_1, \alpha_2, \ldots, \alpha_n$. For this we take the lattice L to be $\mathbb{Z}^n$, embedded in $\mathbb{R}^{n+1}$ by

$$(m_1, m_2, \ldots, m_n) \mapsto \left(m_1, m_2, \ldots, m_n, c\sum_{i=1}^{n} m_i\alpha_i'\right);$$

here c is a large constant and α_i' is a good rational approximation to α_i. The first basis vector of a reduced basis of L will give rise to integers $m_1, m_2, \ldots, m_n$ that are not too large such that $\sum_{i=1}^{n} m_i\alpha_i$ is very small.

Applying this to $\alpha_i = \alpha^{i-1}$ we see that our algorithm can be used to test a given real number α for algebraicity, and to determine its irreducible polynomial. Taking for α a zero of a polynomial $f \in \mathbb{Z}[X]$, $f \neq 0$, and generalizing the algorithm to *complex* α, one finds in this way an irreducible factor of f in $\mathbb{Z}[X]$. It is likely that this yields actually a polynomial-time algorithm to factor f in $\mathbb{Q}[X]$, an algorithm that is different from the p-adic method described in Sect. 3.

In a similar way we can test given real numbers $\alpha, \beta, \gamma, \ldots$ for algebraic dependence, taking the α_i to be the monomials in $\alpha, \beta, \gamma, \ldots$ up to a given degree.

2. Factors and Lattices

In this section we denote by p a prime number and by k a positive integer. We write $\mathbb{Z}/p^k\mathbb{Z}$ for the ring of integers modulo p^k, and $\mathbb{F}_p$ for the field $\mathbb{Z}/p\mathbb{Z}$. For $g=\sum_i a_iX^i\in\mathbb{Z}[X]$ we denote by $(g \bmod p^k)$ the polynomial $\sum_i (a_i \bmod p^k)X^i\in(\mathbb{Z}/p^k\mathbb{Z})[X]$.

We fix a polynomial $f\in\mathbb{Z}[X]$ of degree n, with $n>0$, and a polynomial $h\in\mathbb{Z}[X]$ that has the following properties:

(2.1) $$h \text{ has leading coefficient } 1,$$

(2.2) $$(h \bmod p^k) \text{ divides } (f \bmod p^k) \text{ in } (\mathbb{Z}/p^k\mathbb{Z})[X],$$

(2.3) $$(h \bmod p) \text{ is irreducible in } \mathbb{F}_p[X],$$

(2.4) $$(h \bmod p)^2 \text{ does not divide } (f \bmod p) \text{ in } \mathbb{F}_p[X].$$

We put $l=\deg(h)$; so $0<l\leqq n$.

(2.5) **Proposition.** *The polynomial f has an irreducible factor h_0 in $\mathbb{Z}[X]$ for which $(h \bmod p)$ divides $(h_0 \bmod p)$, and this factor is uniquely determined up to sign. Further, if g divides f in $\mathbb{Z}[X]$, then the following three assertions are equivalent:*

(i) *$(h \bmod p)$ divides $(g \bmod p)$ in $\mathbb{F}_p[X]$,*

(ii) *$(h \bmod p^k)$ divides $(g \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$,*

(iii) *h_0 divides g in $\mathbb{Z}[X]$.*

In particular $(h \bmod p^k)$ divides $(h_0 \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$.

Proof. The existence of h_0 follows from (2.2) and (2.3), and the uniqueness, up to ± 1, from (2.4). The implications (ii) $\Rightarrow$ (i) and (iii) $\Rightarrow$ (i) are obvious. Now assume (i); we prove (iii) and (ii). From (i) and (2.4) it follows that $(h \bmod p)$ does not divide $(f/g \bmod p)$ in $\mathbb{F}_p[X]$. Therefore h_0 does not divide f/g in $\mathbb{Z}[X]$, so it must divide g. This proves (iii). By (2.3) the polynomials $(h \bmod p)$ and $(f/g \bmod p)$ are relatively prime in $\mathbb{F}_p[X]$, so in $\mathbb{F}_p[X]$ we have

$$(\lambda_1 \bmod p)\cdot(h \bmod p)+(\mu_1 \bmod p)\cdot(f/g \bmod p)=1$$

for certain $\lambda_1,\mu_1\in\mathbb{Z}[X]$. Therefore $\lambda_1h+\mu_1 f/g=1-pv_1$ for some $v_1\in\mathbb{Z}[X]$. Multiplying this by $1+pv_1+p^2v_1^2+\ldots+p^{k-1}v_1^{k-1}$ and by g we obtain

$$\lambda_2h+\mu_2f\equiv g \bmod p^k\mathbb{Z}[X]$$

for certain $\lambda_2,\mu_2\in\mathbb{Z}[X]$. Since the left hand side, when taken modulo p^k, is divisible by $(h \bmod p^k)$, the same is true for the right hand side. This proves (ii).

The final assertion of (2.5) follows if we take $g=h_0$. This proves (2.5).

(2.6) In the remainder of this section we fix an integer m with $m\geqq l$, and we let L be the collection of all polynomials in $\mathbb{Z}[X]$ of degree $\leqq m$ that, when taken modulo p^k, are divisible by $(h \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$. This is a subset of the $(m+1)$-dimensional real vector space $\mathbb{R}+\mathbb{R}\cdot X+\ldots+\mathbb{R}\cdot X^m$. This vector space is identified with $\mathbb{R}^{m+1}$ by identifying $\sum_{i=0}^m a_iX^i$ with $(a_0,a_1,\ldots,a_m)$. Notice that the length $\left|\sum_{i=0}^m a_iX^i\right|$ of a polynomial, as defined in the introduction, is equal to the ordinary Euclidean length of $(a_0, a_1, \ldots, a_m)$. It is easy to see that L is a lattice in $\mathbb{R}^{m+1}$ and, using (2.1), that a basis of L is given by

$$\{p^k X^i : 0 \leqq i < l\} \cup \{hX^j : 0 \leqq j \leqq m-l\}.$$

From (1.1) it follows that $d(L) = p^{kl}$.

In the following proposition h_0 is as in (2.5).

(2.7) **Proposition.** *Let $b \in L$ satisfy*

$$p^{kl} > |f|^m \cdot |b|^n. \tag{2.8}$$

Then b is divisible by h_0 in $\mathbb{Z}[X]$, and in particular $\gcd(f, b) \neq 1$.

Remark. A weaker version of (2.7), which could also be used to obtain a polynomial-time factoring algorithm for polynomials, asserts that $\gcd(f, b) \neq 1$ under the same conditions. The proof of this version is less complicated than the proof given below, see [8, Theorem 2].

Proof of (2.7). We may assume that $b \neq 0$. Let $g = \gcd(f, b)$. By (2.5) it suffices to show that $(h \bmod p)$ divides $(g \bmod p)$. Suppose that this is not the case. Then by (2.3) we have

$$\lambda_3 h + \mu_3 g = 1 - p\nu_3 \tag{2.9}$$

for certain $\lambda_3, \mu_3, \nu_3 \in \mathbb{Z}[X]$. We shall derive a contradiction from this.

Put $e = \deg(g)$ and $m' = \deg(b)$. Clearly $0 \leqq e \leqq m' \leqq m$. We define

$$M = \{\lambda f + \mu b : \lambda, \mu \in \mathbb{Z}[X], \deg(\lambda) < m' - e, \deg(\mu) < n - e\}$$
$$\subset \mathbb{Z} + \mathbb{Z} \cdot X + \ldots + \mathbb{Z} \cdot X^{n+m'-e-1}.$$

Let M' be the projection of M on

$$\mathbb{Z} \cdot X^e + \mathbb{Z} \cdot X^{e+1} + \ldots + \mathbb{Z} \cdot X^{n+m'-e-1}.$$

Suppose that $\lambda f + \mu b$ projects to 0 in M', with λ, μ as in the definition of M. Then $\deg(\lambda f + \mu b) < e$, but g divides $\lambda f + \mu b$, so $\lambda f + \mu b = 0$. From $\lambda \cdot (f/g) = -\mu \cdot (b/g)$ and $\gcd(f/g, b/g) = 1$ it follows that f/g divides μ. But $\deg(\mu) < n - e = \deg(f/g)$, so $\mu = 0$, and therefore also $\lambda = 0$.

This proves that the projections of

$$\{X^i f : 0 \leqq i < m' - e\} \cup \{X^j b : 0 \leqq j < n - e\}$$

on M' are linearly independent. Since these projections span M', it follows that M' is a lattice of rank $n + m' - 2e$. From Hadamard's inequality (1.10) and (2.8) we obtain

$$d(M') \leqq |f|^{m'-e} \cdot |b|^{n-e} \leqq |f|^m \cdot |b|^n < p^{kl}. \tag{2.10}$$

Below we deduce from (2.9) that

$$\{v \in M : \deg(v) < e + l\} \subset p^k \mathbb{Z}[X]. \tag{2.11}$$

Hence, if we choose a basis $b_e, b_{e+1}, \ldots, b_{n+m'-e-1}$ of M' with $\deg(b_j)=j$, see [4, Chap. I, Theorem I.A], then the leading coefficients of $b_e, b_{e+1}, \ldots, b_{e+l-1}$ are divisible by p^k. [Notice that $e+l-1 \leqq n+m'-e-1$ because g divides b and $(h \bmod p)$ divides $(f/g \bmod p)$.] Since $d(M')$ equals the absolute value of the product of the leading coefficients of $b_e, b_{e+1}, \ldots, b_{n+m'-e-1}$ we find that $d(M') \geqq p^{kl}$. Combined with (2.10) this is the desired contradiction.

To prove (2.11), let $v \in M$, $\deg(v) < e+l$. Then g divides v. Multiplying (2.9) by v/g and by $1+pv_3+p^2v_3^2+\ldots+p^{k-1}v_3^{k-1}$ we obtain

(2.12) $$\lambda_4 h + \mu_4 v \equiv v/g \bmod p^k \mathbb{Z}[X]$$

with λ_4, $\mu_4 \in \mathbb{Z}[X]$. From $v \in M$ and $b \in L$ it follows that $(v \bmod p^k)$ is divisible by $(h \bmod p^k)$. So by (2.12) also $(v/g \bmod p^k)$ is divisible by $(h \bmod p^k)$. But $(h \bmod p^k)$ is of degree l with leading coefficient 1, while $(v/g \bmod p^k)$ has degree $< e+l-e=l$. Therefore $v/g \equiv 0 \bmod p^k\mathbb{Z}[X]$, so also $v \equiv 0 \bmod p^k\mathbb{Z}[X]$. This proves (2.11).

This concludes the proof of (2.7).

(2.13) **Proposition.** *Let p, k, f, n, h, l be as at the beginning of this section, h_0 as in (2.5), and m, L as in (2.6). Suppose that $b_1, b_2, \ldots, b_{m+1}$ is a reduced basis for L (see (1.4) and (1.5)), and that*

(2.14) $$p^{kl} > 2^{mn/2} \binom{2m}{m}^{n/2} |f|^{m+n}.$$

Then we have $\deg(h_0) \leqq m$ if and only if

(2.15) $$|b_1| < (p^{kl}/|f|^m)^{1/n}.$$

Proof. The "if"-part is immediate from (2.7), since $\deg(b_1) \leqq m$. To prove the "only if"-part, assume that $\deg(h_0) \leqq m$. Then $h_0 \in L$ by (2.5), and $|h_0| \leqq \binom{2m}{m}^{1/2} \cdot |f|$ by a result of Mignotte [10; cf. 7, Exercise 4.6.2.20]. Applying (1.11) to $x=h_0$ we find that $|b_1| \leqq 2^{m/2} \cdot |h_0| \leqq 2^{m/2} \cdot \binom{2m}{m}^{1/2} \cdot |f|$. By (2.14) this implies (2.15). This proves (2.13).

(2.16) **Proposition.** *Let the notation and the hypotheses be the same as in (2.13), and assume in addition that there exists an index $j \in \{1, 2, \ldots, m+1\}$ for which*

(2.17) $$|b_j| < (p^{kl}/|f|^m)^{1/n}.$$

Let t be the largest such j. Then we have

$$\deg(h_0) = m+1-t,$$

$$h_0 = \gcd(b_1, b_2, \ldots, b_t),$$

and (2.17) holds for all j with $1 \leqq j \leqq t$.

Proof. Let $J = \{j \in \{1, 2, \ldots, m+1\}: (2.17) \text{ holds}\}$. From (2.7) we know that h_0 divides b_j for every $j \in J$. Hence if we put

$$h_1 = \gcd(\{b_j : j \in J\})$$

then h_0 divides h_1. Each $b_j, j \in J$, is divisible by h_1 and has degree $\leq m$, so belongs to

$$\mathbb{Z} \cdot h_1 + \mathbb{Z} \cdot h_1 X + \ldots + \mathbb{Z} \cdot h_1 X^{m - \deg(h_1)}.$$

Since the b_j are linearly independent this implies that

$$\#J \leq m + 1 - \deg(h_1). \tag{2.18}$$

By the result of Mignotte used in the proof of (2.13) we have $|h_0 X^i| = |h_0| \leq \binom{2m}{m}^{1/2} \cdot |f|$ for all $i \geq 0$. For $i = 0, 1, \ldots, m - \deg(h_0)$ we have $h_0 X^i \in L$, so from (1.12) we obtain

$$|b_j| \leq 2^{m/2} \cdot \binom{2m}{m}^{1/2} \cdot |f|$$

for $1 \leq j \leq m + 1 - \deg(h_0)$. By (2.14), this implies that

$$\{1, 2, \ldots, m + 1 - \deg(h_0)\} \subset J. \tag{2.19}$$

From (2.18), (2.19) and the fact that h_0 divides h_1 we now see that equality must hold in (2.18) and (2.19), and that

$$\deg(h_0) = \deg(h_1) = m + 1 - t, \qquad J = \{1, 2, \ldots, t\}.$$

It remains to prove that h_0 is equal to h_1, up to sign, and for this it suffices to check that h_1 is primitive. Choose $j \in J$, and let d_j be the content of b_j. Then b_j/d_j is divisible by h_0, and $h_0 \in L$, so $b_j/d_j \in L$. But b_j belongs to a basis for L, so $d_j = 1$ and b_j is primitive, and the same is true for the factor h_1 of b_j. This finishes the proof of (2.16).

Remark. If $t = 1$ then we see from (2.16) that b_1 is an irreducible factor of f, and that no gcd computation is necessary.

Remark. From the proofs of (2.13) and (2.16) we see that (2.14) may be replaced by

$$p^{kl} > \beta^n \gamma^n |f|^m,$$

where $\beta = \max\{|b_j|/|b_i^*| : 1 \leq j \leq i \leq m + 1\}$ [cf. (1.14)] and where γ is such that $|g| \leq \gamma$ for every factor g of f in $\mathbb{Z}[X]$ with $\deg(g) \leq m$.

3. Description of the Algorithm

Denote by f a primitive polynomial in $\mathbb{Z}[X]$ of degree n, with $n > 0$. In this section we describe an algorithm that factors f into irreducible factors in $\mathbb{Z}[X]$. We begin with two auxiliary algorithms.

(3.1) Suppose that, in addition to f and n, a prime number p, a positive integer k and a polynomial $h \in \mathbb{Z}[X]$ are given satisfying (2.1), (2.2), (2.3), and (2.4). Assume that the coefficients of h are reduced modulo p^k, so

$$|h|^2 \leq 1 + l p^{2k},$$

where $l = \deg(h)$. Let further an integer $m \geqq l$ be given, and assume that inequality (2.14) is satisfied:

$$p^{kl} > 2^{mn/2} \cdot \binom{2m}{m}^{n/2} \cdot |f|^{m+n}.$$

We describe an algorithm that decides whether $\deg(h_0) \leqq m$, with h_0 as in (2.5), and determines h_0 if indeed $\deg(h_0) \leqq m$.

Let L be the lattice defined in (2.6), with basis

$$\{p^k X^i : 0 \leqq i < l\} \cup \{hX^j : 0 \leqq j \leqq m-l\}.$$

Applying algorithm (1.15) we find a *reduced* basis $b_1, b_2, \ldots, b_{m+1}$ for L. If $|b_1| \geqq (p^{kl}/|f|^m)^{1/n}$ then by (2.13) we have $\deg(h_0) > m$, and the algorithm stops. If $|b_1| < (p^{kl}/|f|^m)^{1/n}$ then by (2.13) and (2.16) we have $\deg(h_0) \leqq m$ and

$$h_0 = \gcd(b_1, b_2, \ldots, b_t)$$

with t as in (2.16). This gcd can be calculated by repeated application of the subresultant algorithm described in [7, Sect. 4.6.1]. This finishes the description of algorithm (3.1).

(3.2) **Proposition.** *The number of arithmetic operations needed by algorithm* (3.1) *is* $O(m^4 k \log p)$, *and the integers on which these operations are performed each have binary length* $O(mk \log p)$.

Proof. We apply (1.26) with $m+1$ in the role of n and with $B = 1 + lp^{2k}$. From $l \leqq n$ and (2.14) we see that $m = O(k \log p)$, so $\log l < l \leqq m$ implies that $\log B = O(k \log p)$. This leads to the estimates in (3.2). It is straightforward to verify that the gcd computation at the end satisfies the same estimates. This proves (3.2).

(3.3) Next suppose that, in addition to f and n, a prime number p and a polynomial $h \in \mathbb{Z}[X]$ are given such that (2.1), (2.2), (2.3), and (2.4) are satisfied with k replaced by 1. Assume that the coefficients of h are reduced modulo p. We describe an algorithm that determines h_0, the irreducible factor of f for which $(h \bmod p)$ divides $(h_0 \bmod p)$, cf. (2.5).

Write $l = \deg(h)$. If $l = n$ then $h_0 = f$, and the algorithm stops. Let now $l < n$. We first calculate the least positive integer k for which (2.14) holds with m replaced by $n-1$:

$$p^{kl} > 2^{(n-1)n/2} \cdot \binom{2(n-1)}{n-1}^{n/2} \cdot |f|^{2n-1}.$$

Next we modify h, without changing $(h \bmod p)$, in such a way that (2.2) holds for the value of k just calculated, in addition to (2.1), (2.3), and (2.4). This can be accomplished by the use of Hensel's lemma, see [7, Exercise 4.6.2.22; 14; 15; 13]. We may assume that the coefficients of h are reduced modulo p^k.

Let u be the greatest integer for which $l \leqq (n-1)/2^u$. We perform algorithm (3.1) for each of the values $m = [(n-1)/2^u]$, $[(n-1)/2^{u-1}]$, $\ldots$, $[(n-1)/2]$, $n-1$ in succession, with $[x]$ denoting the greatest integer $\leqq x$; but we stop as soon as for one of these values of m algorithm (3.1) succeeds in determining h_0. If this does not occur for any m in the sequence then $\deg(h_0) > n-1$, so $h_0 = f$ and we stop. This finishes the description of algorithm (3.3).

(3.4) **Proposition.** *Denote by $m_0 = \deg(h_0)$ the degree of the irreducible factor h_0 of f that is found by algorithm* (3.3). *Then the number of arithmetic operations needed by algorithm* (3.3) *is $O(m_0(n^5 + n^4 \log|f| + n^3 \log p))$, and the integers on which these operations are performed each have binary length $O(n^3 + n^2 \log|f| + n \log p)$.*

Proof. From

$$p^{k-1} \leqq p^{(k-1)l} \leqq 2^{(n-1)n/2} \binom{2(n-1)}{n-1}^{n/2} |f|^{2n-1}$$

it follows that

$$k \log p = (k-1) \log p + \log p = O(n^2 + n \log|f| + \log p).$$

Let m_1 be the largest value of m for which algorithm (3.1) is performed. From the choice of values for m it follows that $m_1 < 2m_0$, and that every other value for m that is tried is of the form $[m_1/2^i]$, with $i \geqq 1$. Therefore we have $\sum m^4 = O(m_0^4)$. Using (3.2) we conclude that the total number of arithmetic operations needed by the applications of algorithm (3.1) is $O(m_0^4 k \log p)$, which is

$$O(m_0^4(n^2 + n \log|f| + \log p)),$$

and that the integers involved each have binary length $O(m_1 k \log p)$, which is

$$O(m_0(n^2 + n \log|f| + \log p)).$$

With some care it can be shown that the same estimates are valid for a suitable version of Hensel's lemma. But it is simpler, and sufficient for our purpose, to replace the above estimates by the estimates stated in (3.4), using that $m_0 \leqq n$; then a very crude estimate for Hensel's lemma will do. The straightforward verification is left to the reader. This proves (3.4).

(3.5) We now describe an algorithm that factors a given primitive polynomial $f \in \mathbb{Z}[X]$ of degree $n > 0$ into irreducible factors in $\mathbb{Z}[X]$.

The first step is to calculate the resultant $R(f, f')$ of f and its derivative f', using the subresultant algorithm [7, Sect. 4.6.1]. If $R(f, f') = 0$ then f and f' have a greatest common divisor g in $\mathbb{Z}[X]$ of positive degree, and g is also calculated by the subresultant algorithm. This case will be discussed at the end of the algorithm. Assume now that $R(f, f') \neq 0$.

In the second step we determine the smallest prime number p not dividing $R(f, f')$, and we decompose $(f \bmod p)$ into irreducible factors in $\mathbb{F}_p[X]$ by means of Berlekamp's algorithm [7, Sect. 4.6.2]. Notice that $R(f, f')$ is, up to sign, equal to the product of the leading coefficient of f and the discriminant of f. So $R(f, f') \not\equiv 0 \bmod p$ implies that $(f \bmod p)$ still has degree n, and that it has no multiple factors in $\mathbb{F}_p[X]$. Therefore (2.4) is valid for every irreducible factor $(h \bmod p)$ of $(f \bmod p)$ in $\mathbb{F}_p[X]$.

In the third step we assume that we know a decomposition $f = f_1 f_2$ in $\mathbb{Z}[X]$ such that the complete factorizations of f_1 in $\mathbb{Z}[X]$ and $(f_2 \bmod p)$ in $\mathbb{F}_p[X]$ are known. At the start we can take $f_1 = 1$, $f_2 = f$. In this situation we proceed as follows. If $f_2 = \pm 1$ then $f = \pm f_1$ is completely factored in $\mathbb{Z}[X]$, and the algorithm stops. Suppose now that f_2 has positive degree, and choose an irreducible factor

$(h \bmod p)$ of $(f_2 \bmod p)$ in $\mathbb{F}_p[X]$. We may assume that the coefficients of h are reduced modulo p and that h has leading coefficient 1. Then we are in the situation described at the start of algorithm (3.3), with f_2 in the role of f, and we use that algorithm to find the irreducible factor h_0 of f_2 in $\mathbb{Z}[X]$ for which $(h \bmod p)$ divides $(h_0 \bmod p)$. We now replace f_1 and f_2 by $f_1 h_0$ and f_2/h_0, respectively, and from the list of irreducible factors of $(f_2 \bmod p)$ we delete those that divide $(h_0 \bmod p)$. After this we return to the beginning of the third step.

This finishes the description of the algorithm in the case that $R(f, f') \neq 0$. Suppose now that $R(f, f') = 0$, let g be the gcd of f and f' in $\mathbb{Z}[X]$, and put $f_0 = f/g$. Then f_0 has no multiple factors in $\mathbb{Z}[X]$, so $R(f_0, f_0') \neq 0$, and we can factor f_0 using the main part of the algorithm. Since each irreducible factor of g in $\mathbb{Z}[X]$ divides f_0 we can now complete the factorization of $f = f_0 g$ by a few trial divisions. This finishes the description of algorithm (3.5).

(3.6) **Theorem.** *The above algorithm factors any primitive polynomial $f \in \mathbb{Z}[X]$ of positive degree n into irreducible factors in $\mathbb{Z}[X]$. The number of arithmetic operations needed by the algorithm is $O(n^6 + n^5 \log|f|)$, and the integers on which these operations are performed each have binary length $O(n^3 + n^2 \log|f|)$. Here $|f|$ is as defined in the introduction.*

Using the classical algorithms for the arithmetic operations we now arrive at the bound $O(n^{12} + n^9(\log|f|)^3)$ for the number of bit operations that was announced in the introduction. This can be reduced to $O(n^{9+\varepsilon} + n^{7+\varepsilon}(\log|f|)^{2+\varepsilon})$, for every $\varepsilon > 0$, if we employ fast multiplication techniques.

Proof of (3.6). The correctness of the algorithm is clear from its description. To prove the estimates we first assume that $R(f, f') \neq 0$. We begin by deriving an upper bound for p. Since p is the least prime not dividing $R(f, f')$ we have

$$\prod_{q<p,\, q \text{ prime}} q \leqq |R(f, f')|. \tag{3.7}$$

It is not difficult to prove that there is a positive constant A such that

$$\prod_{q<p,\, q \text{ prime}} q > e^{Ap} \tag{3.8}$$

for all $p > 2$, see [6, Sect. 22.2]; by [12] we can take $A = 0.84$ for $p > 101$. From Hadamard's inequality (1.10) we easily obtain

$$|R(f, f')| \leqq n^n |f|^{2n-1}.$$

Combining this with (3.7) and (3.8) we conclude that

$$p < (n \log n + (2n-1) \log|f|)/A \tag{3.9}$$

or $p = 2$. Therefore the terms involving $\log p$ in proposition (3.4) are absorbed by the other terms.

The call of algorithm (3.3) in the third step requires $O(m_0 \cdot (n^5 + n^4 \log|f_2|))$ arithmetic operations, by (3.4), where m_0 is the degree of the factor h_0 that is found. Since f_2 divides f, Mignotte's theorem [10; cf. 7, Exercise 4.6.2.20] that was used in the proof of (2.13) implies that $\log|f_2| = O(n + \log|f|)$. Further the sum $\sum m_0$ of the degrees of the irreducible factors of f is clearly equal to n. We conclude that the total number of arithmetic operations needed by the applications of (3.3) is $O(n^6 + n^5 \log|f|)$. By (3.4), the integers involved in (3.3) each have binary length $O(n^3 + n^2 \log|f|)$.

We must now show that the other parts of the algorithm satisfy the same estimates. For the subresultant algorithm in the first step and the remainder of the third step this is entirely straightforward and left to the reader. We consider the second step.

Write P for the right hand side of (3.9). Then p can be found with $O(P)$ arithmetic operations on integers of binary length $O(P)$; here one can apply [11] to generate a table of prime numbers $< P$, or alternatively use a table of squarefree numbers, which is easier to generate. From $p < P$ it also follows that Berlekamp's algorithm satisfies the estimates stated in the theorem, see [7, Sect. 4.6.2].

Finally, let $R(f, f') = 0$, and $f_0 = f/\gcd(f, f')$ as in the algorithm. Since f_0 divides f, Mignotte's theorem again implies that $\log|f_0| = O(n + \log|f|)$. The theorem now follows easily by applying the preceding case to f_0.

This finishes the proof of (3.6).

(3.10) For the algorithms described in this section the precise choice of the basis reduction algorithm is irrelevant, as long as it satisfies the estimates of proposition (1.26). A few simplifications are possible if the algorithm explained in Sect. 1 is used. Specifically, the gcd computation at the end of algorithm (3.1) can be avoided. To see this, assume that $m_0 = \deg(h_0)$ is indeed $\leq m$. We claim that h_0 occurs as b_1 in the course of the basis reduction algorithm. Namely, by (1.37) it will happen at a certain moment that $b_1, b_2, \ldots, b_{m_0+1}$ form a reduced basis for the lattice of rank $m_0 + 1$ spanned by $\{p^k X^i : 0 \leq i < l\} \cup \{hX^j : 0 \leq j \leq m_0 - l\}$. At that moment, we have $h_0 = b_1$, by (2.13) and (2.16), applied with m_0 in the role of m. A similar argument shows that in algorithm (3.3) one can simply try the values $m = l, l+1, \ldots, n-1$ in succession, until h_0 is found.

Acknowledgements are due to J. J. M. Cuppen for permission to include his improvement of our basis reduction algorithm in Sect. 1.

References

1. Adleman, L.M., Odlyzko, A.M.: Irreducibility testing and factorization of polynomials, to appear. Extended abstract: Proc. 22nd Annual IEEE Symp. Found. Comp. Sci., pp. 409–418 (1981)
2. Brentjes, A.J.: Multi-dimensional continued fraction algorithms. Mathematical Centre Tracts 145. Amsterdam: Mathematisch Centrum 1981
3. Cantor, D.G.: Irreducible polynomials with integral coefficients have succinct certificates. J. Algorithms **2**, 385–392 (1981)
4. Cassels, J.W.S.: An introduction to the geometry of numbers. Berlin, Heidelberg, New York: Springer 1971
5. Ferguson, H.R.P., Forcade, R.W.: Generalization of the Euclidean algorithm for real numbers to all dimensions higher than two. Bull. Am. Math. Soc. **1**, 912–914 (1979)
6. Hardy, G.H., Wright, E.M.: An introduction to the theory of numbers. Oxford: Oxford University Press 1979
7. Knuth, D.E.: The art of computer programming, Vol. 2, Seminumerical algorithms. Reading: Addison-Wesley 1981

8. Lenstra, A.K.: Lattices and factorization of polynomials, Report IW 190/81. Amsterdam: Mathematisch Centrum 1981
9. Lenstra, H.W., Jr.: Integer programming with a fixed number of variables. Math. Oper. Res. (to appear)
10. Mignotte, M.: An inequality about factors of polynomials. Math. Comp. **28**, 1153–1157 (1974)
11. Pritchard, P.: A sublinear additive sieve for finding prime numbers. Comm. ACM **24**, 18–23 (1981)
12. Barkley Rosser, J., Schoenfeld, L.: Approximate formulas for some functions of prime numbers. Ill. J. Math. **6**, 64–94 (1962)
13. Yun, D.Y.Y.: The Hensel lemma in algebraic manipulation. Cambridge: MIT 1974; reprint: New York: Garland 1980
14. Zassenhaus, H.: On Hensel factorization. I. J. Number Theory **1**, 291–311 (1969)
15. Zassenhaus, H.: A remark on the Hensel factorization method. Math. Comp. **32**, 287–292 (1978)
16. Zassenhaus, H.: A new polynomial factorization algorithm (unpublished manuscript, 1981)

Received July 11, 1982

Lecture 2
LLL Algorithm

Lecturer: Oded Regev
Scribe: Eyal Kaplan

In this lecture[1] we describe an approximation algorithm to the Shortest Vector Problem (SVP). This algorithm, developed in 1982 by A. K. Lenstra, H. W. Lenstra, Jr. and L. Lovasz, usually called the **LLL** algorithm, gives a $(\frac{2}{\sqrt{3}})^n$ approximation ratio, where n is the dimension of the lattice. In many of the applications, this algorithm is applied for a constant n; in such cases, we obtain a constant approximation factor.

In 1801, Gauss gave an algorithm that can be viewed as an algorithm for solving SVP in two dimensions. The LLL algorithm is, in some way, a generalization of Gauss's algorithm to higher dimensions. In 1987, Schnorr presented an improved algorithm for the SVP. This improved algorithm obtains an approximation factor that is slightly subexponential, namely $2^{O(n(\log\log n)^2/\log n)}$.

The LLL algorithm has many applications in diverse fields of computer science. Some of these will be described in the following lectures. Here is a brief description of some of these applications.

1. Factoring polynomials over the integers or the rational numbers. For example, given $x^2 - 1$ factor it into $x + 1$ and $x - 1$.

2. Finding the minimal polynomial of an algebraic number given to a good enough approximation. For example, given 1.414213 output $x^2 - 2 = 0$ and given 0.645751 output $x^2 + 4x - 3 = 0$.

3. Finding integer relations. A set of real numbers $\{x_1, \ldots, x_n\}$ is said to posses an integer relation if there exist integers $\{a_1, \ldots, a_n\}$ such that $a_1x_1 + \ldots + a_nx_n = 0$, with not all $a_i = 0$. As an example, try to find an integer relation among $\arctan(1) \approx 0.785398$, $\arctan(\frac{1}{5}) \approx 0.197395$, and $\arctan(\frac{1}{239}) \approx 0.004184$. It turns that an integer relation exists:

$$\arctan(1) - 4\arctan(1/5) + \arctan(1/239) = 0$$

(this equality is known as Machin's formula).

4. Integer Programming. This is a well-known **NP**-complete problem. Using LLL, one can obtain a polynomial time solution to integer programming with a fixed number of variables.

5. Approximation to the Closest Vector Problem (CVP), as well as other lattice problems.

6. Various applications in cryptanalysis (i.e., breaking cryptographic protocols). For example, there are many attacks on knapsack based cryptographic systems. Moreover, there are some more recent attacks on some special cases of RSA such as the low public exponent attack.

For simplicity, we describe the LLL algorithm for full-rank lattices; it is easy to remove this restriction. Moreover, our description only applies to the ℓ_2 norm. Extensions to other norms are known.

Let us now turn to describe LLL. The exposition is divided into three stages.

1. Define an LLL reduced basis.

2. Present an algorithm to find such a basis.

3. Analyze its running time.

[1]Last updated: 2013/2/5

1 Reduced basis

We first recall the Gram-Schmidt orthogonalization process.

DEFINITION 1 *Given n linearly independent vectors $b_1, \dots, b_n \in \mathbb{R}^n$, the* ***Gram-Schmidt orthogonalization*** *of $b_1, \dots, b_n$ is defined by $\tilde{b_i} = b_i - \sum_{j=1}^{i-1} \mu_{i,j}\tilde{b_j}$, where $\mu_{i,j} = \frac{\langle b_i, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle}$.*

DEFINITION 2 *A basis $B = \{b_1, \dots, b_n\} \in \mathbb{R}^n$ is a* ***δ-LLL Reduced Basis*** *if the following holds:*

1. $\forall 1 \leq i \leq n, j < i. |\mu_{i,j}| \leq \frac{1}{2}$,
2. $\forall 1 \leq i < n.\ \delta\|\tilde{b_i}\|^2 \leq \|\mu_{i+1,i}\tilde{b_i} + \tilde{b}_{i+1}\|^2$.

REMARK 1 It is always possible to transform a basis to a reduced basis. Actually, this is what the LLL algorithm does.

REMARK 2 It is helpful to consider the case $\delta = \frac{3}{4}$. The algorithm works with any $\frac{1}{4} < \delta < 1$.

REMARK 3 The second property in Definition 2 can be written as:

$$\delta\|\tilde{b_i}\|^2 \leq \|\mu_{i+1,i}\tilde{b_i} + \tilde{b}_{i+1}\|^2 = \mu_{i+1,i}^2\|\tilde{b_i}\|^2 + \|\tilde{b}_{i+1}\|^2$$

where the second equality follows since $\tilde{b_i}$ and $\tilde{b}_{i+1}$ are orthogonal. It follows that

$$\|\tilde{b}_{i+1}\|^2 \geq (\delta - \mu_{i+1,i}^2)\|\tilde{b_i}\|^2 \geq (\delta - \frac{1}{4})\|\tilde{b_i}\|^2$$

Put this way, the second property reads "$\tilde{b}_{i+1}$ is not much shorter than $\tilde{b_i}$".

To better understand this definition, consider the orthonormal basis obtained by normalization the Gram-Schmidt vectors $\tilde{b}_1, \dots, \tilde{b}_n$. In this basis, B can be written as

$$\begin{pmatrix} \|\tilde{b_1}\| & * & \cdots & * \\ 0 & \|\tilde{b_2}\| & \cdots & * \\ \vdots & & \ddots & \vdots \\ & & & * \\ 0 & \cdots & & \|\tilde{b_n}\| \end{pmatrix}$$

where column i shows the coordinates of b_i in this orthonormal basis. The first condition in the definition of an LLL-reduced basis guarantees that the absolute value of any off-diagonal element is at most half the value written in the diagonal element on the same row. This can be written as

$$\begin{pmatrix} \|\tilde{b_1}\| & \leq \frac{1}{2}\|\tilde{b_1}\| & \cdots & \leq \frac{1}{2}\|\tilde{b_1}\| \\ 0 & \|\tilde{b_2}\| & \cdots & \leq \frac{1}{2}\|\tilde{b_2}\| \\ \vdots & & \ddots & \vdots \\ & & & \leq \frac{1}{2}\|\tilde{b}_{n-1}\| \\ 0 & \cdots & & \|\tilde{b_n}\| \end{pmatrix}$$

where $\le \frac{1}{2}\|\tilde{b_j}\|$ indicates that the absolute value of this coordinate is at most $\frac{1}{2}\|\tilde{b_j}\|$. For the second property, consider the 2×2 submatrix of the above matrix, with the upper left entry indexed at (i,i).

$$\begin{pmatrix} \|\tilde{b_i}\| & \mu_{i+1,i}\|\tilde{b_i}\| \\ 0 & \|\tilde{b}_{i+1}\| \end{pmatrix}$$

Then the second property requires that the second column of this matrix is almost as long as its first column. Let us mention that in Schnorr's improvement to the LLL algorithm, this second property is replaced with some condition on $k \times k$ submatrices for some $k > 2$.

One important property of LLL-reduced basis is that its first vector is relatively short, as shown in the next claim.

CLAIM 1 *Let $b_1, \ldots, b_n \in \mathbb{R}^n$ be a δ-LLL-reduced basis. Then $\|b_1\| \le (\frac{2}{\sqrt{4\delta-1}})^{n-1}\lambda_1(\mathcal{L})$.*

REMARK 4 For $\delta = \frac{3}{4}$ this gives $\|b_1\| \le 2^{(n-1)/2}\lambda_1(\mathcal{L})$.

PROOF: Since for any basis $b_1, \ldots, b_n$, $\lambda_1(\mathcal{L}) \ge \min_i \|\tilde{b_i}\|$, we get that

$$\|\tilde{b_n}\|^2 \ge (\delta - \frac{1}{4})\|\tilde{b}_{n-1}\|^2 \ge \ldots \ge (\delta - \frac{1}{4})^{n-1}\|\tilde{b_1}\|^2 = (\delta - \frac{1}{4})^{n-1}\|b_1\|^2$$

where the last equality follows by the definition $\tilde{b_1} = b_1$. Then, for any i,

$$\|\tilde{b_1}\| \le \left(\delta - \frac{1}{4}\right)^{-(i-1)/2}\|\tilde{b_i}\| \le \left(\delta - \frac{1}{4}\right)^{-(n-1)/2}\|\tilde{b_i}\|.$$

Hence,

$$\|b_1\| \le \left(\delta - \frac{1}{4}\right)^{-(n-1)/2}\min_i \|\tilde{b_i}\| \le \left(\delta - \frac{1}{4}\right)^{-(n-1)/2} \cdot \lambda_1(\mathcal{L})$$

□

REMARK 5 LLL-reduced bases have many other good properties; some are mentioned in the homework.

Claim 1 provides us with an approximation to the SVP problem. Assuming we can generate a δ-LLL-reduced basis from our input basis, we can then return b_1 as our answer. For $\delta = 3/4$ we obtain a $2^{(n-1)/2}$ approximation. In what follows, we describe how to transform an arbitrary basis into a δ-LLL-reduced one.

2 The LLL Algorithm

INPUT: Lattice basis $b_1, \ldots, b_n \in \mathbb{Z}^n$
OUTPUT: δ-LLL-reduced basis for $\mathcal{L}(B)$

```
Start: compute $\tilde{b_1}, \ldots, \tilde{b_n}$
Reduction Step:
  **for** $i$ = 2 to $n$ **do**
    **for** $j = i - 1$ to 1 **do**
      $b_i \leftarrow b_i - c_{i,j}b_j$ where $c_{i,j} = \lceil \langle b_i, \tilde{b_j}\rangle / \langle \tilde{b_j}, \tilde{b_j}\rangle \rfloor$
```

```
Swap Step:
    if $\exists i$ s.t. $\delta\|\tilde{b}_i\|^2 > \|\mu_{i+1,i}\tilde{b}_i + \tilde{b}_{i+1}\|^2$ then
        $b_i \leftrightarrow b_{i+1}$
        goto start
Output $b_1, \ldots, b_n$
```

REMARK 6 We use $\lceil \cdot \rfloor$ to denote rounding to the nearest integer, e.g., $\lceil 3.3 \rfloor = 3$, $\lceil 3.8 \rfloor = 4$.

Let us make some important observations on this procedure. It is easy to see that the swap step takes care of the second property of an LLL-reduced basis. Indeed, if the algorithm ever terminates, then its output must satisfy the second property. The reduction step takes care of the first property. In order to see this, first notice that throughout the reduction step, the Gram-Schmidt basis does not change (hence the vectors $\tilde{b}_1, \ldots, \tilde{b}_n$ need not be recomputed). This holds since we only perform column operations of the form $b_i \leftarrow b_i + ab_j$ for $i > j$ and $a \in \mathbb{Z}$. Such operations to not change the Gram-Schmidt orthogonalization. In the ith iteration of the outer loop, the reduction step makes sure that the projection of b_i on $\tilde{b}_j$ for any $j < i$ is at most $\frac{1}{2}\|\tilde{b}_j\|$. It does so by subtracting from column i the right integer multiple of column j such that the jth coordinate becomes at most $\frac{1}{2}\|\tilde{b}_j\|$ in absolute value. Notice that it is crucial that the inner loop goes from $i-1$ *down to* 1.

To demonstrate the reduction step, let us write B in the orthonormal basis obtained by normalizing the Gram-Schmidt vectors. Consider, for example, the ith iteration of the outer loop and the $j = 2$ iteration of the inner loop. Then at this point, the matrix B looks like

$$\begin{pmatrix}
\|\tilde{b}_1\| & \le \frac{1}{2}\|\tilde{b}_1\| & \le \frac{1}{2}\|\tilde{b}_1\| & \cdots & * & * & \cdots \\
0 & \|\tilde{b}_2\| & \le \frac{1}{2}\|\tilde{b}_2\| & \cdots & * & * & \cdots \\
0 & & \|\tilde{b}_3\| & \cdots & \le \frac{1}{2}\|\tilde{b}_3\| & * & \cdots \\
\vdots & & \ddots & & & \vdots & \\
 & & & & \le \frac{1}{2}\|\tilde{b}_{i-1}\| & * & \\
0 & \cdots & & & \|\tilde{b}_i\| & * & \cdots \\
 & & & & 0 & \|\tilde{b}_{i+1}\| & \cdots \\
\vdots & & & & \vdots & & \ddots
\end{pmatrix}$$

At this iteration, we subtract some integer multiple of the second column from column i to make the second entry in the ith column at most $\frac{1}{2}\|\tilde{b}_2\|$ in absolute value. Similarly, in the last iteration of the inner loop, we subtract some integer multiple of the first column from column i.

LEMMA 3 (CORRECTNESS) *If the LLL procedure described above ever terminates, then its output is a δ-LLL-reduced basis for the lattice spanned by the input basis $b_1, \ldots, b_n$.*

PROOF: We need to prove that the output of the LLL algorithm is a basis for $\mathcal{L}(B)$ that satisfies both properties of a δ-LLL-reduced basis. The second property of a δ-LLL-reduced basis is enforced by the check during the swap step. The reason that the output of the algorithm is indeed a basis for $\mathcal{L}(B)$, is that we only perform column operations of the form $b_i \leftarrow b_i + ab_j$ for $i \neq j$, and $a \in \mathbb{Z}$.

We next show that after the reduction step, $b_1, \ldots, b_n$ satisfy $|\mu_{i,j}| \le \frac{1}{2}$, for all $i > j$. First, notice that throughout the reduction step, the Gram-Schmidt basis does not change. Now, consider some

$i > j$, and consider the jth iteration of the inner loop in the ith iteration of the outer loop. Then $|\mu_{i,j}|$ can be written as

$$|\mu_{i,j}| = \Big|\frac{\langle b_i - c_{i,j} \cdot b_j, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle}\Big| = \Big|\frac{\langle b_i, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle} - \Big\lceil \frac{\langle b_i, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle} \Big\rfloor \cdot \frac{\langle b_j, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle}\Big| \le \frac{1}{2}$$

where the first equality follows from the definition of the reduction step and the last inequality follows from the fact that $\langle b_j, \tilde{b_j}\rangle = \langle \tilde{b_j}, \tilde{b_j}\rangle$. □

3 Analyzing the Running Time

Our analysis consists of two steps. First, we bound the number of iterations. Second, we bound the running time of a single iteration.

We show that the overall running time of the algorithm is polynomial in the input size. A rough lower bound on the latter is given by $M := \max\{n, \log(\max_i \|b_i\|)\}$ (because each of the n vectors requires at least one bit to represents and a vector of norm r requires at least $\log r$ bits to represent). In the following, we show that the running time of the algorithm is polynomial in M.

LEMMA 4 *The number of iterations is polynomial in M.*

PROOF: Our first step is to define a function mapping a lattice basis to some positive number. This function can be thought of as a 'potential function'.

DEFINITION 5 *Let $B = \{b_1, \dots, b_n\}$ be a lattice basis. The potential of B, denoted $\mathcal{D}_B$, is defined by*

$$\prod_{i=1}^{n} \|\tilde{b_i}\|^{n-i+1} = \prod_{i=1}^{n} \|\tilde{b_1}\|\|\tilde{b_2}\| \cdots \|\tilde{b_i}\| = \prod_{i=1}^{n} \mathcal{D}_{B,i}$$

where $\mathcal{D}_{B,i} := \det \Lambda_i$ and Λ_i is defined as the lattice spanned by $b_1, \dots, b_i$

REMARK 7 Notice that more weight is given to the first vectors.

Our aim is to show that the initial value of $\mathcal{D}_B$ is not too large, and that it decays quickly. Since $\|\tilde{b_i}\| \le \|b_i\|$, the initial value of $\mathcal{D}_B$ can be bounded from above by $(\max_i \|b_i\|)^{n(n+1)/2}$. Note that the logarithm of this value is polynomial in M.

During the reduction step, $\mathcal{D}_B$ does not change, because the Gram-Schmidt basis does not change. Now consider the swap step. Suppose that b_i is swapped with b_{i+1}. For all $k \neq i$, Λ_k does not change, and so $\mathcal{D}_{B,k}$ does not change; only $\mathcal{D}_{B,i}$ changes. Let Λ_i', $\mathcal{D}_{B,i}'$ denote the new values of Λ_i and $\mathcal{D}_{B,i}$, respectively. We have that

$$\begin{aligned}
\frac{\mathcal{D}_{B,i}'}{\mathcal{D}_{B,i}} &= \frac{\det \Lambda_i'}{\det \Lambda_i} \\
&= \frac{\det \mathcal{L}(b_1, \dots, b_{i-1}, b_{i+1})}{\det \mathcal{L}(b_1, \dots, b_i)} \\
&= \frac{(\prod_{j=1}^{i-1} \|\tilde{b_j}\|) \|\mu_{i+1,i}\tilde{b_i} + \tilde{b}_{i+1}\|}{\prod_{j=1}^{i} \|\tilde{b_j}\|} \\
&= \frac{\|\mu_{i+1,i}\tilde{b_i} + \tilde{b}_{i+1}\|}{\|\tilde{b_i}\|} < \sqrt{\delta}
\end{aligned}$$

where the last inequality follows from the condition in the swap step.

As shown above, in each iteration, $\mathcal{D}_B$ decreases by a multiplicative factor, $\sqrt{\delta}$. Let $\mathcal{D}_{B,0}$ be the initial value of $\mathcal{D}_B$. Since $\mathcal{D}_B$ is a nonzero integer, and in particular at least 1, this means that we can bound from above the number of iterations by

$$\log_{\frac{1}{\sqrt{\delta}}} \mathcal{D}_{B,0} = \frac{\log \mathcal{D}_{B,0}}{\log \frac{1}{\sqrt{\delta}}} \leq \frac{1}{\log \frac{1}{\sqrt{\delta}}} \cdot \frac{n(n+1)}{2} \log(\max_i \|b_i\|).$$

For any constant $\delta < 1$, this is polynomial in M. $\square$

REMARK 8 A somewhat tedious calculation shows that even for $\delta = \frac{1}{4} + (\frac{3}{4})^{\frac{n}{n-1}}$, which is closer to 1 than any constant, the running time is polynomial. For such δ the approximation factor is $(\frac{2}{\sqrt{3}})^n$. This approximation factor is essentially the best one can obtain with the LLL algorithm. For better approximation factors, one needs to apply Schnorr's algorithm.

LEMMA 6 *The running time of each iteration is polynomial in M.*

PROOF: It is not difficult to see that in each iteration we perform only a polynomial number of arithmetic operations (i.e., additions, multiplications, etc.). Hence, in the rest of the proof, it is enough to show that the numbers that arise in each iteration can be represented using a polynomial number of bits.

To demonstrate why this is necessary, consider a repeated squaring algorithm that given a number x, squares it n times. Even though the number of arithmetic operations is only n, the number of bits required to represent the resulting numbers quickly grows to $2^{O(n)}$. Hence, the actual running time of the algorithm (measured in bit operations) is *exponential* in n.

We establish the bound on numbers arising during an iteration using two claims. The first concerns the Gram-Schmidt vectors $\tilde{b}_1, \ldots, \tilde{b}_n$, which are somewhat simpler to bound, as they do not change during the reduction step. The second concerns the basis vectors $b_1, \ldots, b_n$.

CLAIM 2 *The Gram-Schmidt vectors $\tilde{b}_1, \ldots, \tilde{b}_n$ can be computed in polynomial time in M. Moreover, for every $1 \leq i \leq n$, we have that $\mathcal{D}_B^2 \tilde{b}_i \in \mathbb{Z}^n$ and that $\|\tilde{b}_i\| \leq \mathcal{D}_B^2$.*

REMARK 9 Notice that these two properties of the Gram-Schmidt vectors imply that they can be represented in space polynomial in M. Indeed, the bound on the norm implies that each coordinate of $\tilde{b}_i$ contains a number of absolute value at most $\mathcal{D}_B^2$. Moreover, since $\mathcal{D}_B^2 \tilde{b}_i \in \mathbb{Z}^n$ we know that the denominators cannot be larger than $\mathcal{D}_B^2$. Hence, each coordinate requires at most $O(\log \mathcal{D}_B)$ bits to represent and there are n^2 of them. Since the initial value of $\log \mathcal{D}_B$ is polynomial in M and later on it can only decrease, we obtain that the Gram-Schmidt vectors can be represented in space polynomial in M.

PROOF: The calculation of the Gram-Schmidt basis may be performed as follows. Since $\tilde{b}_i - b_i \in \text{span}(b_1, \ldots, b_{i-1})$, we can write $\tilde{b}_i = b_i + \sum_{j=1}^{i-1} a_j b_j$, for some $a_1, \ldots, a_{i-1} \in \mathbb{R}$. We are looking for $a_1, \ldots, a_{i-1}$ such that $\tilde{b}_i$ is orthogonal to each of $b_1, \ldots, b_{i-1}$. For any $1 \leq l \leq i-1$, $\langle \tilde{b}_i, b_l \rangle = 0$ can be written as

$$\langle \tilde{b}_i, b_l \rangle = \langle b_i + \sum_{j=1}^{i-1} a_j b_j, b_l \rangle = \langle b_i, b_l \rangle + a_1 \langle b_1, b_l \rangle + a_2 \langle b_2, b_l \rangle + \ldots + a_{i-1} \langle b_{i-1}, b_l \rangle = 0.$$

Hence, we obtain the following system of $i-1$ linear equations in $i-1$ variables:

$$\begin{aligned} a_1\langle b_1, b_1\rangle + a_2\langle b_2, b_1\rangle + \ldots + a_{i-1}\langle b_{i-1}, b_1\rangle &= -\langle b_i, b_1\rangle \\ a_1\langle b_1, b_2\rangle + a_2\langle b_2, b_2\rangle + \ldots + a_{i-1}\langle b_{i-1}, b_2\rangle &= -\langle b_i, b_2\rangle \\ &\vdots \\ a_1\langle b_1, b_{i-1}\rangle + a_2\langle b_2, b_{i-1}\rangle + \ldots + a_{i-1}\langle b_{i-1}, b_{i-1}\rangle &= -\langle b_i, b_{i-1}\rangle. \end{aligned}$$

It is possible to solve such a system in polynomial time.

For the second part of the claim, notice that using Cramer's rule we can write

$$a_j = \frac{\det(\text{some integer matrix})}{\det\begin{pmatrix} \langle b_1, b_1\rangle & \ldots & \langle b_{i-1}, b_1\rangle \\ \vdots & \ddots & \vdots \\ \langle b_1, b_{i-1}\rangle & \ldots & \langle b_{i-1}, b_{i-1}\rangle \end{pmatrix}} = \frac{\text{some integer}}{\det B_{i-1}^T B_{i-1}} = \frac{\text{some integer}}{(\det \Lambda_{i-1})^2}.$$

Hence $\tilde{b_i} = b_i + \sum_{j=1}^{i-1} a_j b_j$ for some rational numbers a_j whose denominator is $(\det \Lambda_{i-1})^2$. This implies that $\mathcal{D}_{B,i}^2 \tilde{b_i}$ and in particular also $\mathcal{D}_B^2 \tilde{b_i}$ are integer vectors.

Now we show that the norm of the $\tilde{b_i}$'s is not too large. By Definition 5,

$$\mathcal{D}_{B,i} = (\prod_{j=1}^{i-1} \|\tilde{b_j}\|) \cdot \|\tilde{b_i}\|$$

and so

$$\|\tilde{b_i}\| = \frac{\mathcal{D}_{B,i}}{\prod_{j=1}^{i-1} \|\tilde{b_j}\|} \le \mathcal{D}_{B,i} \prod_{j=1}^{i-1} \mathcal{D}_{B,j}^2 \le \mathcal{D}_B^2$$

where the first inequality follows since $\|\tilde{b_j}\| \ge \frac{1}{\mathcal{D}_{B,j}^2}$. $\square$

In the next claim we show that the basis vectors b_i do not become too large. This is necessary since these basis vectors change during the reduction step (and in fact, it is possible for vectors to become longer by the reduction step). We first bound the length of each b_i after the ith iteration of the outer loop is done (i.e., once vector b_i is reduced). We then bound the length of b_i *during* the ith iteration of the outer loop. For this we use the observation that to vector b_i we only add vectors b_j for $j < i$; these vectors are already reduced and hence our first bound applies.

CLAIM 3 *All vectors b_i appearing during an iteration can be represented using poly(M) bits.*

PROOF: First, we show that after the reduction step, the length of the b_i's is not too large. For each $1 \le i \le n$,

$$\|b_i\|^2 = \|\tilde{b_i}\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|\tilde{b_j}\|^2 \le \mathcal{D}_B^4 + \frac{n}{4} \cdot \mathcal{D}_B^4 \le n\mathcal{D}_B^4$$

The first equality holds because $\tilde{b}_1, \ldots, \tilde{b}_n$ are orthogonal. The first inequality follows from the bound on $\tilde{b}_1, \ldots, \tilde{b}_n$ proven in Claim 2, and using the fact that $|\mu_{i,j}| \le \frac{1}{2}$.

Our bound on the norm implies that each coordinate contains an integer of size at most $\sqrt{n}\mathcal{D}_B^2$. For an integer vector, this means that it can be represented in $\log(\sqrt{n}\mathcal{D}_B^2)$ bits. Our b_i's remain integer vectors throughout the procedure – they are such as inputs, and we change their values by adding integers. This means that after the reduction step, we can represent the b_i's in $\text{poly}(M)$ space.

Lastly, we need to show that *during* the reduction step, the b_i's are not too large. Consider a vector b_i, that is manipulated in the inner loop of the reduction step.

$$|c_{i,j}| = \Big|\Big\lceil \frac{\langle b_i, \tilde{b_j}\rangle}{\langle \tilde{b_j}, \tilde{b_j}\rangle} \Big\rfloor\Big| \leq \frac{\|b_i\|\|\tilde{b_j}\|}{\|\tilde{b_j}\|^2} + 1 = \frac{\|b_i\|}{\|\tilde{b_j}\|} + 1 \leq \frac{\|b_i\|}{1/\mathcal{D}_B^2} + 1 \leq 2\mathcal{D}_B^2\|b_i\|$$

where the first inequality follows by applying Cauchy-Schwartz and using the definition of the rounding operator, and the second inequality uses Claim 2. Therefore,

$$\begin{aligned}
\|b_i - c_{i,j}b_j\| &\leq \|b_i\| + |c_{i,j}|\|b_j\| \\
&\leq (1 + 2\mathcal{D}_B^2\|b_j\|)\|b_i\| \\
&\leq (1 + 2\mathcal{D}_B^2\sqrt{n}\mathcal{D}_B^2)\|b_i\| \\
&\leq (4n\mathcal{D}_B)^4\|b_i\|
\end{aligned}$$

where the first inequality follows by the triangle inequality, the second inequality by plugging in the bound for $|c_{i,j}|$, and the third inequality by plugging in the bound on the length of $\|b_j\|$ after the reduction step. Indeed, during the reduction step of b_i, vectors b_j, for $j < i$, have already finished their reduction step, so we can use this bound. After at most n iterations of the inner loop, the norm of b_i has increased by a factor of at most $(4n\mathcal{D}_B)^{4n}$. This is of course representable in $\text{poly}(M)$ size. □

By Claims 2 and 3 we have, that it is possible to represent the numbers in a polynomial number of bits. This, together with the fact that in each iteration we perform a polynomial number of arithmetic operations, proves the lemma. □

REMARK 10 The only place where we used that $|\mu_{i,j}| \leq \frac{1}{2}$ for *all* $j < i$ was in the proof of Claim 3. For the rest of the proof, the weaker condition that $|\mu_{i+1,i}| \leq \frac{1}{2}$ for all i is enough. This suggests that we might improve the running time by performing the reduction step only on pairs of consecutive vectors so as to obtain the weaker condition. The number of iterations in this modified algorithm is still polynomial, since all of our arguments above hold. However, it is not clear if this modified algorithm still runs in polynomial time because Claim 3 does not seem to hold.

We combine Lemma 4 with Lemma 6 to conclude that the running time of the LLL algorithm is polynomial in the input size. This completes our analysis of LLL.

Open questions

The worst-case behavior of LLL and its generalization BKZ are reasonably well understood [1], and it turns out that the analysis above is tight in the worst-case. However, according to extensive experiments done by Gama and Nguyen [2], for "typical" lattices, the LLL algorithm (and its generalizations) appear to behave much better than the worst-case analysis suggests. Although the dependence on the dimension is still exponential, the base of the exponent is much smaller than the $(\delta - 1/4)^{-1/2}$ we obtained above. Explaining this phenomenon, even heuristically, is still an open question. Another outstanding open question is to improve on LLL and its generalizations for special families of lattices (e.g., rotations of $\mathbb{Z}^n$ or so-called ideal lattices).

References

[1] M. Ajtai. Optimal lower bounds for the korkine-zolotareff parameters of a lattice and for Schnorr's algorithm for the shortest vector problem. *Theory of Computing*, 4(2):21–51, 2008.

[2] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.

CSE 206A: Lattice Algorithms and Applications Winter 2012

3: The LLL Algorithm

Instructor: *Daniele Micciancio* UCSD CSE

No efficient algorithm is known to find the shortest vector in a lattice (in arbitrary dimension), or even just computing its length λ_1. A central tool in the algorithmic study of lattices (and their applications) is the LLL algorithm of Lenstra, Lenstra and Lovasz. The LLL algorithm runs in polynomial time and finds an approximate solution $\mathbf{x}$ to the shortest vector problem, in the sense that the length of the solution $\mathbf{x}$ found by the algorithm is at most $\gamma \cdot \lambda_1$, for some approximation factor γ. The approximation factor $\gamma = 2^{O(n)}$ achieved by LLL is exponential in the dimension of the lattice. Later in the course, we will study polynomial time algorithms that achieve (slightly) better factors. Still, the approximate solutions found by LLL are enough in many applications. We design and analyze the LLL algorithm in two steps:

(1) We first define a notion of "reduced" basis, and show that the first vector of a reduced basis is an approximately shortest vector in the lattice.
(2) Next, we give an efficient algorithm to compute a reduced basis for any lattice.

1. Reduced basis

Remember the Gram-Schmidt orthogonalization process:

$$\mathbf{b}_i^* \;=\; \mathbf{b}_i - \textstyle\sum_{j<i} \mu_{i,j}\mathbf{b}_j^* \qquad \text{where} \qquad \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^*\rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^*\rangle}$$

Define the orthogonal projection operations π_i from $\mathbb{R}^m$ onto $\sum_{j\geq i}\mathbb{R}\mathbf{b}_j^*$by

$$\pi_i(\mathbf{x}) = \sum_{j=i}^{n} \frac{\langle \mathbf{x}, \mathbf{b}_j^*\rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^*\rangle}\mathbf{b}_j^*.$$

Notice that the Gram-Schmidt orthogonalized vectors can be expressed as $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$.

We can now define LLL-reduced basis. For reasons that will be clear in the running time analysis of the algorithm, we introduce a real parameter $1/4 < \delta < 1$ and define LLL-reduced basis with respect to δ.

Definition 1. A basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m\times n}$ is δ-LLL reduced if:

- $|\mu_{i,j}| \leq \frac{1}{2}$ for all $i > j$
- for any any pair of consecutive vectors $\mathbf{b}_i, \mathbf{b}_{i+1}$, we have
$$\delta\|\pi_i(\mathbf{b}_i)\|^2 \leq \|\pi_i(\mathbf{b}_{i+1})\|^2.$$

The first condition (usually called "size reduction") is easy to achieve using an integer variant of the Gram-Schmidt orthogonalization procedure, and it is discussed in the next section. In order to understand the second condition it is useful to consider the case when $i = 1$ and $\delta = 1$. For $i = 1$, the projection π_1 is just the identity function (over the linear span of the lattice) and the condition becomes simply $\|\mathbf{b}_1\| \leq \|\mathbf{b}_2\|$, i.e., the first two vectors in an LLL reduced basis are sorted in order of nondecreasing length. (Introducing a factor $\delta < 1$ relaxes the nondecreasing condition to allow a small decrease δ in the norm.) For

$i > 1$, the LLL reduced basis definition requires a similar condition to hold for the projected basis $\pi_i(\mathbf{B})$.

Another geometric interpretation of the second condition is the following. Notice that

$$\|\pi_i(\mathbf{b}_{i+1})\|^2 = \|\mathbf{b}^*_{i+1} + \mu_{i+1,i}\mathbf{b}^*_i\|^2 = \|\mathbf{b}^*_{i+1}\|^2 + \|\mu_{i+1,i}\mathbf{b}^*_i\|^2 = \|\mathbf{b}^*_{i+1}\|^2 + (\mu_{i+1,i})^2\|\mathbf{b}^*_i\|^2.$$

So, the second condition in the definition of LLL-reduced basis can be equivalently rewritten as

$$(\delta - \mu^2_{i+1,i})\|\mathbf{b}^*_i\|^2 \leq \|\mathbf{b}^*_{i+1}\|^2.$$

So, although the Gram-Schmidt vectors $\mathbf{b}^*_i$ can get shorter and shorter, their length cannot decrease too quickly. Specifically, for any $1/4 < \delta < 1$, if we set $\alpha = \frac{1}{\delta - \frac{1}{4}}$, then

$$\|\mathbf{b}^*_i\|^2 \leq \alpha\|\mathbf{b}^*_{i+1}\|^2. \tag{1.1}$$

For example, if $\delta = 3/4$, then $\alpha = 2$ and each $\|\mathbf{b}^*_{i+1}\|^2$ is least $\frac{1}{2}\|\mathbf{b}^*_i\|^2$. Using (1.1) repeatedly, we get

$$\|\mathbf{b}^*_1\|^2 \leq \alpha^{i-1}\|\mathbf{b}^*_i\|^2 \leq \alpha^{n-1}\|\mathbf{b}^*_i\|^2. \tag{1.2}$$

Since this is true for all $i = 1, \ldots, n$, the first vector in an LLL reduced basis satisfies

$$\|\mathbf{b}_1\| \leq \alpha^{(n-1)/2} \min \|\mathbf{b}^*_i\| \leq \alpha^{(n-1)/2}\lambda_1$$

where we have used the lower bound $\lambda_1 \geq \min_i \|\mathbf{b}^*_i\|$ on the length of the shortest vector in a lattice. In particular, if $\delta = 3/4$, the first vector in an LLL reduced basis is a $\gamma = 2^{(n-1)/2}$ approximate solution to SVP. Simiarly, one can also show that the set of vectors in an LLL reduced basis are a solution to the approximate SIVP.

Exercise 2. Prove that if $\mathbf{B}$ is a δ-LLL reduced basis, then $\max_i \|\mathbf{b}_i\| \leq \alpha^{(n-1)/2}\lambda_n$ where $\alpha = (1 - 1/\delta)^{-1}$.

In many applications, the length of the shortest lattice vector λ_1 is not known, but it can be estimated using Minkowski's theorem $\lambda_1 \leq \sqrt{n}\det(\mathbf{B})^{1/n}$. Combining the bound $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/2}\lambda_1$ with Minkowski's theorem we get $\|\mathbf{b}_1\| \leq \sqrt{n}\alpha^{(n-1)/2}\det(\mathbf{B})^{1/n}$. A stronger bound can be obtained relating the length of $\mathbf{b}_1$ in an LLL reduced basis directly to the determinant of the lattice as follows. Take the product of (1.2) for $i = 1, \ldots, n$, to get

$$\|\mathbf{b}_1\|^n \leq \prod_i \alpha^{(i-1)/2}\|\mathbf{b}^*_i\| = \alpha^{n(n-1)/4}\det(\mathbf{B}).$$

So, we have

$$\|\mathbf{b}_1\| \leq \alpha^{(n-1)/4}\det(\mathbf{B}^{1/n}),$$

which can be interpreted as a weak (but algorithmic) version of Minkowski's theorem.

Remark 3. Even if Minkowski's bound can be efficiently computed given a lattice basis, no efficient algorithm is known to find lattice vectors achieving the bound, even approximately for approximation factors that are significantly better than exponental in the dimension of the lattice.

Our analysis of LLL reduced basis is summarized in the following theorem.

Theorem 4. *For any $1/4 < \delta \leq 1$, if $\mathbf{B}$ is a δ-LLL reduced basis, then*

- $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/2}\lambda_1$

- $\max_i \|\mathbf{b}_i\| \leq \alpha^{(n-1)/2}\lambda_n$
- $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/4}\det(\mathbf{B})^{1/n}$

where $\alpha = 1/(\delta - 1/4) \geq 4/3$.

In the following sections we give an algorithm to compute a δ-LLL reduced basis for any lattice in time polynomial in the input size and $(1-\delta)^{-1}$. It is not known whether the LLL algorithm runs in polynomial time when $\delta = 1$, which is the value that gives the best results. Still, we can achieve essentially the same result by setting $\delta = 1 - 1/n^c = 1 - o(1)$ and still maintain polynomial running time. This gives polynomial time solutions to SVP and SIVP for exponential approximation factors.

Corollary 5. *There is a polynomial time algorithm that solves SVP and SIVP within approximation factor* $\gamma(n) = (2/\sqrt{3})^n$. *The algorithm also produces nonzero lattice vectors of length at most* $(2/\sqrt{3})^{n/2} \cdot \det(\mathcal{L}(\mathbf{B}))^{1/n}$.

The LLL algorithm is designed to work in the Euclidean norm. Still, since all norms are within a factor n from the Euclidean norm, is also provides solutions to lattice problems in other norms within essentially the same approximation factor.

2. The Nearest Plane Algorithm

The size reduction condition ($|\mu_{i,j}| \leq 1/2$) in the definition of LLL reduced basis can be achieved using an integer variant of the Gram-Schmidt orthogonalization procedure. Both the size reduction condition and the associated algorithm have nice geometric interpretations which we are going to explain first.

It is easy to see that any point in the linear span of a lattice can be written as the sum of a lattice point $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ plus a vector in the fundamental parallelepiped

$$\mathbf{y} \in \mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} : 0 \leq \mathbf{x} < 1\}.$$

Moreover, such a decomposition is unique. In other words, the sets $\mathbf{x} + \mathcal{P}(\mathbf{B})$ (indexed by $\mathbf{x} \in \mathcal{L}(\mathbf{B})$) form a partition of span$(\mathbf{B})$. A subset $S \subseteq \text{span}(\mathbf{B})$ such that $\{\mathbf{x} + S : \mathbf{x} \in \mathcal{L}(\mathbf{B})\}$ form a partition of span$(\mathbf{B})$ is called a *fundamental region* for the lattice, and $\mathcal{P}(\mathbf{B})$ is an example of fundamental region. There are many other examples of interesting fundamental regions. For example, one can consider the *centered* half open parallelepiped

$$\mathcal{C}(\mathbf{B}) = \left\{\mathbf{Bx}\colon -\frac{1}{2} \leq \mathbf{x} < +\frac{1}{2}\right\}.$$

Another important fundamental region is the Voronoi cell of the lattice $\mathcal{V}(\mathbf{B})$, i.e., the set of all points that are closer to the origin than to any other lattice point.[1]

Notice that the partition associated to $\mathcal{P}(\mathbf{B})$ can be easily computed, in the sense that given a target point $\mathbf{t} \in \text{span}(\mathbf{B})$, one can efficiently find the lattice point $\mathbf{Bx}$ such that $\mathbf{t} \in \mathbf{Bx} + \mathcal{P}(\mathbf{B})$. (Just solve $\mathbf{By} = \mathbf{t}$ and round the solution to the lattice point $\mathbf{B}\lfloor \mathbf{y} \rfloor$.) The partition associated to the centered parallepiped $\mathcal{C}(\mathbf{B})$ can also be computed similarly,

[1]In order to get a partition, one needs also to include boundary points, with some care to avoid including the same point in multiple regiones. For example, the norm relation $\|\mathbf{x}\| \leq \|\mathbf{y}\|$ can be extended to a total order by defining $\mathbf{x} < \mathbf{y}$ if and only if $\|\mathbf{x}\| < \|\mathbf{y}\|$ or $\|\mathbf{x}\| = \|\mathbf{y}\|$ and the first nonzero coordinate of $\mathbf{x} - \mathbf{y}$ is negative. Then, the *half-open* Voronoi cell can be defined as the sef of all points $\mathbf{x}$ such that $\mathbf{x} \leq (\mathbf{x} - \mathbf{y})$ for any lattice point $\mathbf{y} \in \mathcal{L}(\mathbf{B})$.

Algorithm 1 Nearest Plane Algorithm. On input a lattice basis $\mathbf{B}$ and a target vector $\mathbf{t}$, output a lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\langle \mathbf{t}-\mathbf{v}, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|^2 \in [-1/2, 1/2)$ for all $i = 1, \ldots, n$.

```
NearestPlane($\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$,$\mathbf{t}$):
if $n = 0$ then return $\mathbf{0}$
else $\mathbf{B}^* \leftarrow$ GramSchmidt($\mathbf{B}$)
    $c \leftarrow \left\lfloor \frac{\langle \mathbf{t}, \mathbf{b}_n^* \rangle}{\|\mathbf{b}_n^*\|^2} \right\rceil$
    return $c\mathbf{b}_n$ + NearestPlane($[\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}], \mathbf{t} - c\mathbf{b}_n$)
```

rounding to the closest integers $\mathbf{B}\lfloor \mathbf{y} \rceil$. On the other hand, the partition associated to the Voronoi cell seems hard to compute: by definition, finding which Voronoi cell $\mathbf{Bx} + \mathcal{V}(\mathbf{B})$ contains a given target point $\mathbf{t}$ is equivalent to finding the lattice point $\mathbf{Bx}$ closest to $\mathbf{t}$, and instance of the CVP.

The size reduction condition in the definition of LLL reduced basis can be easily interpreted as partitioning the space according to still another fundamental region: the orthogonalized centered parallelepiped $\mathcal{C}(\mathbf{B}^*)$, where $\mathbf{B}^*$ is the Gram-Schmidt matrix of $\mathbf{B}$.

Exercise 6. Prove that $\mathcal{C}(\mathbf{B}^*)$ is a fundamental region for lattice $\mathcal{L}(\mathbf{B})$.

The cell $\mathbf{Bx} + \mathcal{C}(\mathbf{B}^*)$ containing a given target $\mathbf{t}$ can be easily found using the Nearest Plane algorithm, a simple variant of the Gram-Schmidt algorithm given as Algorithm 1.

Algorithm 1, on input a rank $n > 0$ lattice $\mathbf{B}$ and a target $\mathbf{t}$ proceeds as follows. Let $\mathbf{B}' = [\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$ be the sublattice generated by the first $n-1$ basis vectors. The lattice $\mathcal{L}(\mathbf{B})$ can be decomposed into hyperplanes of the form

$$\mathcal{L}(\mathbf{B}) = c\mathbf{b}_k + \mathcal{L}(\mathbf{B}') \subset c\mathbf{b}_k^* + \text{span}(\mathbf{B}').$$

The algorithm selects the hyperplane $c = \lfloor \langle \mathbf{t}, \mathbf{b}_k^* \rangle / \|\mathbf{b}_k^*\|^2 \rceil$ closest to the target, and recursively search for a lattice point in $c\mathbf{b}_k + \mathcal{L}(\mathbf{B}')$ close to $\mathbf{t}$, or equivalently, a lattice point in the lower dimensional sublattice $\mathcal{L}(\mathbf{B}')$ close to $\mathbf{t} - c\mathbf{b}_k$. The base case of the algorithm is when the rank of the lattice is reduced to 0 and the only possible output is the origin $\mathbf{0}$.

Lemma 7. *On input a lattice basis and a target vector* $\mathbf{t}$*, Algorithm 1 outputs a lattice vector* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *such that* $\langle \mathbf{t} - \mathbf{v}, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|^2 \in [-1/2, 1/2)$ *for all* $i = 1, \ldots, n$*. In particular, if* $\mathbf{t} \in \textit{span}(\mathbf{B})$*, then*$\mathbf{t} \in \mathbf{v} + \mathcal{C}(\mathbf{B}^*)$*.*

Proof. For the base case, the property is vacuously true. So assume $n > 0$ and that the lemma holds for lower rank lattices. Let $\mathbf{B} = [\mathbf{C}|\mathbf{b}]$ where $\mathbf{C} = [\mathbf{b}_1, \ldots, \mathbf{b}_{\mathbf{n-1}}]$, and notice that $\mathbf{B}^* = [\mathbf{C}^*|\mathbf{b}_n^*]$. By inductive hypothesis, the recursive call returns a lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{C})$ such that $(\mathbf{t} - c\mathbf{b}_n) - \mathbf{v} = \mathbf{C}^*\mathbf{z}$ for some $\mathbf{z}$ such that $z_i \in [-1/2, +1/2]$. The output of the algorithm $\mathbf{v} + c\mathbf{b}_n$ satisfies

$$\langle \mathbf{t} - (\mathbf{v} + c\mathbf{b}_n), \mathbf{b}_i^* \rangle = \langle (\mathbf{t} - c\mathbf{b}_n) - \mathbf{v}, \mathbf{b}_i^* \rangle \in [-1/2, 1/2) \cdot \|\mathbf{b}_n^*\|^2$$

for all $i = 1, \ldots, n-1$ and

$$\langle \mathbf{t} - (\mathbf{v} + c\mathbf{b}_n), \mathbf{b}_n^* \rangle = \langle \mathbf{t}, \mathbf{b}_n^* \rangle - c\langle \mathbf{b}_n, \mathbf{b}_n^* \rangle \in [-1/2, 1/2) \cdot \|\mathbf{b}_n^*\|^2$$

where we have used the fact that $\langle \mathbf{b}_n, \mathbf{b}_n^* \rangle = \|\mathbf{b}_n^*\|^2$. □

Algorithm 2 Size Reduce

```
SizeReduce($\mathbf{B}$):
for i=2 to n
    $\mathbf{x} \leftarrow$ NearestPlane($\mathbf{B}, \mathbf{b}_i - \mathbf{b}_i^*$)
    $\mathbf{b}_i \leftarrow \mathbf{b}_i - \mathbf{B}\mathbf{x}$
output $\mathbf{B}$
```

Algorithm 3 The LLL basis reduction algorithm

```
LLL($\mathbf{B}$,$\delta$):
SizeReduce($\mathbf{B}$)
if $\delta\|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$ for some i
then swap($\mathbf{b}_i$,$\mathbf{b}_{i+1}$); return LLL($\mathbf{B}$,$\delta$)
else return $\mathbf{B}$
```

Remark 8. The fundamental region $\mathcal{C}(\mathbf{B}^*)$ contains a sphere centered in $\mathbf{0}$ of radius $\min_i \|\mathbf{b}_i^*\|/2 \leq \lambda(\mathcal{L}(\mathbf{B}))/2$. Since NearestPlane maps all points in $\mathbf{v} + \mathcal{C}(\mathbf{B}^*)$ to $\mathbf{v} \in \mathcal{L}(\mathbf{B})$, if $\mathbf{t}$ is within distance $\min_i \|\mathbf{b}_i^*\|/2$ from the lattice, then NearestPlane$(\mathbf{B}, \mathbf{t})$ returns the lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ closest to $\mathbf{t}$.

Observe that if $\mathbf{b}_i - \mathbf{b}_i^* \in \mathbf{B}\mathbf{x} + \mathcal{C}(\mathbf{B}^*)$, then $-1/2 \leq \mu_{i,j} < 1/2$ for all $j < i$. So, given a lattice basis $\mathbf{B}$, a size reduced basis for the same lattice can be easily obtained using Algorithm 2.

It is clear that the final $\mathbf{B}$ is a basis for the original lattice because we only executed elementary integer column operations in step 3. Moreover, it is easy to verify that after iteration i, the size reduction condition holds for all $j < i$. Finally, at iteration i, the Gram-Schmidt coefficients $\mu_{i',j}$ with $j < i' < i$ do not change. So, upon termination, the basis is size reduced.

3. The LLL algorithm

The LLL algorithm alternates two steps, aimed at achieving the two properties of an LLL reduced basis. Once we have size-reduced the input basis $\mathbf{B}$, there is only one way $\mathbf{B}$ can fail to be LLL reduced: violate the second condition, i.e., $\delta\|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$ for some index i. If this happens, the algorithm swaps $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. Several pairs might violate the second property. Which one is selected for the swapping does not matter. In the original LLL algorithm i was chosen to be the smallest unordered pair, but any selection is equally good. In fact, one can even swap several disjoint pairs at the same time, leading to a parallel variant of the LLL algorithm. After the swap, the basis is not necessarily size reduced anymore. So, one must repeat the whole process from the reduction step. The LLL algorithm is summarized as Algorithm 3.

Cleary, upon termination the basis is LLL-reduced because it is size reduced and no pairs need to be swapped. So, if the algorithm terminates, then it is correct. We now prove that the algorithm terminates and it is actually polynomial time.

In order to show that the algorithm is polynomial time, we have to prove that the number of iterations is polynomial in the input size, and each iteration takes polynomial time. We first bound the number of iterations.

3.1. **Bounding number of iterations.** We now bound the number of iterations performed by the algorithm, i.e., we analyze the maximum number of swaps that can occur. This is accomplished by associating a positive integer to the basis $\mathbf{B}$, and showing that each time we swap two vectors this integer decreases by at least a constant factor.

Remember the definition of determinant

$$\det(\mathbf{B}) = \prod \|\mathbf{b}_i^*\| = \sqrt{\det(\mathbf{B}^\top \mathbf{B})}.$$

From the second formula it is clear that if $\mathbf{B}$ is an integer matrix, then the square of the determinant is an integer.

Lemma 9. *For every integer basis* $\mathbf{B} \in \mathbb{Z}^{m \times n}$*, we have* $\det(\mathcal{L}(\mathbf{B}))^2 \in \mathbb{Z}$

Therefore, we can associate to the basis $\mathbf{B}$ the following positive integer

$$\mathcal{D} = \prod_{k=1}^{n} det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_k))^2 \in \mathbb{Z}$$

We want to show that $\mathcal{D}$ decreases at least by a factor δ at each iteration. First we will show that Size Reduction does not change $\mathcal{D}$. Then we will show that each swap decreases $\mathcal{D}$ at least by δ.

To prove that Size Reduction doesn't change $\mathcal{D}$ we remember that SizeReduce does not affect the $\mathbf{b}_i^*$'s. Since $\mathcal{D}$ can be expressed as a function of the $\mathbf{b}_i^*$'s, the potential $\mathcal{D}$ is unchanged by the size reduction operation.

At this point we need to look at the effect that a swap has on $\mathcal{D}$. Let us look at the effect of a single swap say between $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. Let $\mathcal{D}$ be the integer associated to the basis B before a swap, and $\mathcal{D}'$ the corresponding integer after the swap.

Notice that for all $j \neq i$ the lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j)$ is not changed by the swap. To prove this look at the two cases $j < i$ and $j > i$. When $j < i$ then there is no change in the basis $[\mathbf{b}_1, \cdots, \mathbf{b}_j]$, so the value of $\det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j))$ remains the same. On the other hand, if $j > i$ the only change is that two basis vectors in $[\mathbf{b}_1, \cdots, \mathbf{b}_j]$ are swapped, so the lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j)$ does not change and the determinant $\det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j))$ stays also the same.

So, the only factor in $\mathcal{D}$ that is affected by the swap is the one corresponding to lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_i)$. Here we are replacing the last vector $\mathbf{b}_i$ by $\mathbf{b}_{i+1}$. Therefore

$$\frac{\mathcal{D}}{\mathcal{D}'} = \frac{\prod_{j \le i} \|\mathbf{b}_j^*\|^2}{(\prod_{j<i} \|\mathbf{b}_j^*\|^2) \cdot \|\pi_i(\mathbf{b}_{i+1})\|^2} = \frac{\|\pi_i(\mathbf{b}_i)\|^2}{\|\pi_i(\mathbf{b}_{i+1})\|^2} \ge \frac{1}{\delta}$$

because swaps are performed only when $\|\pi_i(\mathbf{b}_{i+1})\|^2 < \delta \|\pi_i(\mathbf{b}_i)\|^2$.

This proves that

$$\mathcal{D}' \le \delta \mathcal{D}$$

and by induction on n,

$$\mathcal{D}^{(n)} \le \delta^n \mathcal{D}$$

where $\mathcal{D}$ is the value associated to the initial basis and $\mathcal{D}^{(n)}$ is the value after n iterations. Since $\mathcal{D}$ is a positive integer, $\mathcal{D} \ge 1$ and $(\frac{1}{\delta})^n \le \mathcal{D}$ or equivalently

$$n \le \log_{\frac{1}{\delta}} \mathcal{D} \tag{3.1}$$

This proves an upper bound on the number of iterations as a function of the initial value of $\mathcal{D}$. Since $\mathcal{D}$ is computable in polynomial time from the input basis, then its size must be polynomial in the input size. An estimate of how big $\mathcal{D}$ is can be easily obtained using Hadamard inequality.

3.2. **Bounding the numbers.** We proved that the number of iterations is bounded by a polynomial in the input size. In order to bound the running time of the algorithm we still need to show that each iteration also takes polynomial time. The number of arithmetic operations performed at each iteration is clearly polynomial. So, in order to prove a polynomial bound on the running time we only need to show that the size of the numbers involved in the entire computation also is polynomially bounded.

The LLL algorithm uses rational numbers, so we need to bound both the precision required by this number and their magnitude. In the analysis of the Gram-Schmidt algorithm we have already shown that the denominators in the $\mu_{i,j}$ coefficients must divide D_j. Notice that $\mathcal{D} = \prod D_j$ and therefore all entries in $\mu_{i,j}$ and $\mathbf{b}_i^*$ can be written as integers divided by $\mathcal{D}$. By definition, after size reduction, the $\mu_{i,j}$ are at most $1/2$ in absolute value, so their bit-size is bounded by $\log \mathcal{D}$. We now bound the length of the vectors. Using $D_i = \prod_{j=1}^{i} \|\mathbf{b}_i^*\|^2$, we get

$$\|\mathbf{b}_i^*\|^2 = \frac{D_i}{D_{i-1}} \leq D_i \leq \mathcal{D}.$$

Finally,

$$\|\mathbf{b}_i\|^2 = \|\mathbf{b}_i^*\|^2 + \sum_{j<i} \mu_{i,j}^2 \|\mathbf{b}_j^*\|^2 \leq \mathcal{D} + (n/4)\mathcal{D} \leq n\mathcal{D}.$$

So, all numerators and denominators of the numbers occurring in the execution of the algorithm have bit-size polynomial in $\log \mathcal{D}$.

4. A Simple Application in Number Theory

We give a simple application of LLL to algorithmic number theory: showing that we can *efficiently* write any prime $p \equiv 1 \pmod 4$ as the sum of two squares. Remember the proof that any such prime is the sum of two squares: all vectors $[a, b]^\top$ in the lattice

$$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ i & p \end{bmatrix}$$

have the property that $a^2 + b^2$ is a multiple of p. So if we can find a nonzero lattice vector of squared norm less than $2p$, it must be $a^2 + b^2 = p$. Minwoski's theorem assures us that such short vectors exist. The question is: how can we find it? Answer: using LLL!

Run the LLL algorithm on the lattice basis, to obtain a reduced basis $\mathbf{b}_1, \mathbf{b}_2$ for the same lattice. We know, from Theorem 4 that $\|\mathbf{b}_1\| \leq \alpha^{1/4} \det(\mathbf{B})^{1/2}$. Squaring, and using $\det(\mathbf{B}) = p$, we get $\|\mathbf{b}_1\|^2 \leq \sqrt{\alpha} p < 2p$ as desired, provided $\delta > 3/4$.

5. The Closest Vector Problem

The LLL algorithm can be used also to solve the approximate CVP. No new algorithm is needed: we will show that preprocessing the lattice basis using the LLL algorithm and then applying the NearestPlane algorithm to the target vector produces approximate solutions to CVP within essentially the same approximation factor as the LLL algorithm for SVP. (See Algorithm 4.)

Theorem 10. *Algorithm 4 solves CVP within a factor* $\gamma(n) = (2/\sqrt{3})^n$

Algorithm 4 Approximate CVP algorithm

```
ApproximateCVP($\mathbf{B}$,$\mathbf{t}$) :
$\mathbf{B} \leftarrow$ LLL($\mathbf{B}$)
return NearestPlane($\mathbf{B}$, $\mathbf{t}$)
```

Proof. The algorithm runs in polynomial time because it involves just two polynomial time computations. Proving that the algorithm is correct requires to look again into the details of LLL basis reduction and the Nearest Plane algorithm. We want to prove that if $\mathbf{B}$ is LLL reduced, then NearestPlane solves the approximate CVP. The proof is by induction on the rank of $\mathbf{B}$. The base case when the rank is $n = 0$ is trivial. So, assume $\mathbf{B}$ is an LLL reduced basis of rank $n > 0$. NearestPlane selects a hyperplane index $c = \lfloor \langle \mathbf{t}, \mathbf{b}_n^* \rangle / \|\mathbf{b}_n^*\|^2 \rceil$ and makes a recursive call on input the sublattice $\mathbf{C} = [\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$ and target $\mathbf{t} - c \cdot \mathbf{b}_n$. First of all notice that if $\mathbf{B}$ is LLL reduced, then $\mathbf{C}$ is also LLL reduced. So, we can invoke the inductive hypothesis on the recursive call. Let $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ be a lattice vector closest to $\mathbf{t}$. We distinguish two cases:

Case 1. If $\mathbf{v} \in c\mathbf{b}_n + \mathcal{L}(\mathbf{C})$, then the correctness of the final output follows by induction. Specifically, on input $\mathbf{C}$ and $\mathbf{t} - c\mathbf{b}_n$, the recursive call to NearestPlane returns a lattice point $\mathbf{w} \in \mathcal{L}(\mathbf{C})$ such that

$$\|(\mathbf{t} - c\mathbf{b}_n) - \mathbf{w}\| \leq \gamma(n-1) \cdot \|(\mathbf{t} - c\mathbf{b}_n) - (\mathbf{v} - c\mathbf{b}_n)\| \leq \gamma(n) \cdot \|\mathbf{t} - \mathbf{v}\|.$$

So, Algorithm 4 returns a lattice vector $c\mathbf{b}_n + \mathbf{w}$ at a distance from the target $\mathbf{t}$ which is within a factor $\gamma(n)$ from optimal.

Case 2. Otherwise, it must be $\mathbf{v} \in c'\mathbf{b}_n + \mathcal{L}(\mathbf{C})$ for some $c' \neq c$. Let $\mathbf{t} = \mathbf{t}' + \mathbf{t}''$, where $\mathbf{t}' \in \text{span}(\mathbf{B})$ and $\mathbf{t}'' \perp \text{span}(\mathbf{B})$. Then the distance between $\mathbf{t}$ and $\mathbf{v}$ is at least $\sqrt{\|\mathbf{t}''\|^2 + \|\mathbf{b}_n^*\|^2/4}$. On the other hand, NearestPlane returns a lattice vector $\mathbf{w}$ within distance $\sqrt{\|\mathbf{t}''\|^2 + \sum_i \|\mathbf{b}_i^*\|^2/4}$ from $\mathbf{t}$.Using the property (1.1) of LLL reduces basis we get

$$\frac{\|\mathbf{t} - \mathbf{w}\|}{\|\mathbf{t} - \mathbf{v}\|} \leq \sqrt{\frac{\|\mathbf{t}''\|^2 + \sum_i \|\mathbf{b}_i^*\|^2/4}{\|\mathbf{t}''\|^2 + \|\mathbf{b}_n^*\|^2/4}} \leq \sqrt{\frac{\sum_i \|\mathbf{b}_i^*\|^2/4}{\|\mathbf{b}_n^*\|^2/4}} \leq \sqrt{\sum_i \alpha^{n-i}}.$$

Setting $\gamma = \sqrt{\sum_i \alpha^{n-i}} = \sqrt{(\alpha^n - 1)/(\alpha - 1)} \leq \sqrt{3}(2/\sqrt{3})^n$ concludes the proof.

□

18.409 An Algorithmist's Toolkit Nov. 19, 2009

Lecture 20

Lecturer: Jonathan Kelner

1 Brief Review of Gram-Schmidt and Gauss's Algorithm

Our main task of this lecture is to show a polynomial time algorithm which approximately solves the Shortest Vector Problem (SVP) within a factor of $2^{O(n)}$ for lattices of dimension n. It may seem that such an algorithm with exponential error bound is either obvious or useless. However, the algorithm of Lenstra, Lenstra and Lovász (LLL) is widely regarded as one of the most beautiful algorithms and is strong enough to give some extremely striking results in both theory and practice.

Recall that given a basis $b_1, \ldots, b_n$ for a vector space (no lattices here yet), we can use the Gram-Schmidt process to construct an orthogonal basis $b_1^*, \ldots, b_n^*$ such that $b_1^* = b_1$ and
$b_k^* = b_k - [\text{projection of } b_k \text{ onto span}(b_1, \ldots, b_{k-1})]$ for all $2 \le k \le n$ (note that we do not normalize b_k^*). In particular, we have that for all k:

- $\text{span}(b_1, \ldots, b_k) = \text{span}(b_1^*, \ldots, b_k^*)$,
- $b_k = \sum_{i=1}^{k} \mu_{ki} b_i^*$, and
- $\mu_{kk} = 1$.

The above conditions can be rewritten as $B = MB^*$, where basis vectors are rows of B and B^*, and

$$M = \begin{bmatrix} \mu_{11} & 0 & 0 & \ldots & 0 \\ \mu_{21} & \mu_{22} & 0 & \ldots & 0 \\ \vdots & & \ddots & & \\ \mu_{n1} & \mu_{n2} & \mu_{n3} & \ldots & \mu_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ \mu_{21} & 1 & 0 & \ldots & 0 \\ \vdots & & \ddots & & \\ \mu_{n1} & \mu_{n2} & \mu_{n3} & \ldots & 1 \end{bmatrix}.$$

Obviously $\det(M) = 1$, and thus $\text{vol}(B) = \text{vol}(B^*)$. However, the entries of M are not integers, and thus $L(B) \neq L(B^*)$. We have proved last time that

$$\textit{for any } b \in L,\ ||b|| \ge \min_i\{||b_i^*||\}.$$

We'll use this to prove useful bound for the shortest vector on lattice.

Recall also that last time we saw the Gauss's algorithm which solves SVP for $d = 2$. There are two key ingredients of the algorithm. The first is a definition of "reduced basis" which characterizes the discrete version of bases being orthogonal: namely,

a basis $\{u, v\}$ for a 2-d lattices is said to be *reduced*, if $|u| \le |v|$ and $|u \cdot v| \le \frac{|u|^2}{2}$.

The second is an efficient procedure that produces a reduced basis. The procedure consists of two stages: First is a Euclid-like process which subtracts a multiple of the shorter vector from the longer one to get a vector as short as possible. The second stage is, if the length ordering is broken, we swap the two vectors and repeat, otherwise (i.e., $|u| \le |v|$) the procedure ends. To make the above procedure obviously terminate in polynomial time, we change the termination criterion to be $(1 - \epsilon)|u| \le |v|$. This only gives us a $(1 - \epsilon)$-approximation, but is good enough. The basic idea of LLL algorithm is to generalize Gauss's algorithm to higher dimensions.

2 LLL Algorithm

2.1 Reduced Basis

In order to find a short vector in the lattice, we would like to perform a discrete version of GS procedure. To this end, we need to formalize the notion of being orthogonal in lattice problems. One way to do this is to say that the result of our procedure is "almost orthogonalized" so that doing Gram-Schmidt does not change much.

Definition 1 (Reduced Basis) *Let $\{b_1, \dots, b_n\}$ be a basis for a lattice L and let M be its GS matrix defined in Section 1. $\{b_1, \dots, b_n\}$ is a reduced basis if it meets the following two conditions:*

- *Condition 1: all the non-diagonal entries of M satisfy $|\mu_{ik}| \leq 1/2$.*
- *Condition 2: for each i, $||\pi_{S_i} b_i||^2 \leq \frac{4}{3} ||\pi_{S_i} b_{i+1}||^2$, where S_i is the orthogonal complement of (i.e., the subspace orthogonal to) span$(b_1, \dots, b_{i-1})$, and π_{S_i} is the projection operator to S_i.*

Remark The constant $4/3$ here is to guarantee polynomial-time termination of the algorithm, but the choice of the exact value is somewhat arbitrary. In fact, any number in $(1, 4)$ will do.

Remark Condition 2 is equivalent to $||b_{i+1}^* + \mu_{i+1,i} b_i^*||^2 \geq \frac{3}{4} ||b_i^*||^2$ and one may think it as requiring that the projections of any two successive basis vectors b_i and b_{i+1} onto S_i satisfy a gapped norm ordering condition, analogous to what we did in Gauss's algorithm for 2D case.

2.2 The algorithm

Given $\{b_1, \dots, b_n\}$, the LLL algorithm works as below.

LLL Algorithm for SVP

```
Repeat the following two steps until we have a reduced basis

Step 1: Gauss Reduction
        Compute the GS matrix $M$
    for $i = 1$ to $n$
        for $k = i - 1$ to 1
            $m \leftarrow$ nearest integer to $\mu_{ik}$
            $b_i \leftarrow b_i - m b_k$
        end
    end

Step 2: Swapping
    if exists $i$ s.t. $||\pi_{S_i} b_i||^2 > \frac{4}{3} ||\pi_{S_i} b_{i+1}||^2$
        then swap $b_i$ and $b_{i+1}$
             go to Step 1
```

3 Analysis of LLL Algorithm

The LLL algorithm looks pretty intuitive, but it is not obvious at all that it converges in polynomial number of steps or gives a good answer to SVP. We'll see that it indeed works.

3.1 LLL produces a short vector

We first show that reduced basis gives a short vector.

Claim 2 *If $b_1, \ldots, b_n$ is a reduced basis, then $||b_1|| \leq 2^{\frac{n-1}{2}} \lambda_1(L)$.*

Proof Note that

$$\begin{aligned} ||b_i^*||^2 = ||\pi_{S_i} b_i||^2 &\leq \frac{4}{3}||\pi_{S_i} b_{i+1}||^2 \\ &= \frac{4}{3}||b_{i+1}^* + \mu_{i+1,i} b_i^*||^2 = \frac{4}{3}||b_{i+1}^*||^2 + \frac{4}{3}\mu_{i+1,i}^2 ||b_i^*||^2 \\ &\leq \frac{4}{3}||b_{i+1}^*||^2 + \frac{1}{3}||b_i^*||^2, \end{aligned}$$

which gives $||b_{i+1}^*||^2 \geq \frac{1}{2}||b_i^*||^2$. By induction on i, we have

$$||b_i^*||^2 \geq \frac{1}{2^{i-1}}||b_1^*||^2 = \frac{1}{2^{i-1}}||b_1||^2.$$

Recall that $\forall b \in L$, $||b|| \geq \min_i ||b_i^*||$. Therefore $\lambda_1(L) \geq \min_i ||b_i^*||$, which combined with the inequality above yields

$$||b_1||^2 \leq \min_i \{2^{i-1}||b_i^*||^2\} \leq 2^{n-1} \min_i \{||b_i^*||^2\} \leq 2^{n-1}\lambda_1(L)^2$$

as desired. ■

3.2 Convergence of LLL

Now we show that the LLL algorithm terminates in polynomial time. Note that in each iteration of LLL, Step 1 takes polynomial time and Step 2 takes $O(1)$ times. What we need to show is that we only need to repeat Step 1 and Step 2 a polynomial number of times. To this end, we define a potential function as follows:

$$D(b_1, \ldots, b_n) = \prod_{i=1}^{n} ||b_i^*||^{n-i}.$$

It is clear that Step 1 does not change D since we do not change the Gram-Schmidt basis.

We are going to show that each iteration of Step 2 decreases D by a constant factor. In Step 2, we swap i and $i+1$ only when $||b_i^*||^2 > 4/3||\pi_{S_i} b_{i+1}||^2 \geq 4/3||b_{i+1}^*||^2$. Therefore each swapping decreases D by a factor of at least $2/\sqrt{3}$, as desired.

It is left to show that D can be upper- and lower-bounded. Since $||b_i^*|| \leq ||b_i||$, the initial value of D can be upper bounded by $(\max_i ||b_i||)^{n(n-1)/2}$. On the other hand, we may rewrite D as $\prod_{i=1}^{n} |\det(\Lambda_i)|$, where Λ_i is the lattice spanned by $b_1, \ldots, b_i$. Since we assume that the lattice basis vectors are integer-valued, so D is at least 1.

In sum, the algorithm must terminate in $\log_{2/\sqrt{3}}(\max_i ||b_i||)^{n(n-1)/2} = \text{poly}(n)$ iterations.

4 Application of LLL–Lenstra's Algorithm for Integer Programming

4.1 Applications of LLL

LLL algorithm has many important applications in various fields of computer science. Here are a few (many taken from Regev's notes):

1. Solve integer programming in bounded dimension as we are going to see next.

2. Factor polynomials over the integers or rationals. Note that this problem is harder than the same task but over reals, e.g. it needs to distinguish $x^2 - 1$ from $x^2 - 2$.
3. Given an approximation of an algebraic number, find its minimal polynomial. For example, given 0.645751 outputs $x^2 + 4x - 3$.
4. Find integer relations among a set of numbers. A set of real numbers $\{x_1, \dots, x_n\}$ is said to have an integer relation if there exists a set of integers $\{a_1, \dots, a_n\}$ not identically zero such that $a_1x_1 + \cdots + a_nx_n = 0$. As an example, if we are given $\arctan(1), \arctan(1/5)$ and $\arctan(1/239)$, we should output $\arctan(1) - 4\arctan(1/5) + \arctan(1/239) = 0$. How would you find this just given these numbers as decimals?
5. Approximate to SVP, CVP and some other lattice problems.
6. Break a whole bunch of cryptosystems. For example, RSA with low public exponent and many knapsack based cryptographic systems.
7. Build real life algorithms for some NP-hard problems, e.g. subset sum problem.

4.2 Integer Programming in Bounded Dimension

4.2.1 Linear, Convex and Integer Programming

Consider the following feasibility version of the linear programming problem:

- Linear Programming (feasibility)

  Given: An $m \times n$ matrix A and a vector $b \in \mathbb{R}^n$

  Goal: Find a point $x \in \mathbb{R}^n$ s.t. $Ax \le b$, or determine (with a certificate) that none exists

One can show that other versions, such as the optimization version, are equivalent to feasibility version.
If we relax the searching regions from polytopes to convex bodies, we get convex programming.

- Convex Programming (feasibility)

  Given: A separation oracle for a convex body K and a promise that
  - K is contained in a ball of singly exponential radius R
  - if K is non-empty, it contains a ball of radius r which is at least 1/(singly exponential)

  Goal: Find a point $x \in \mathbb{R}^n$ that belongs to K, or determine (with a certificate) that none exists

Integer programming is the same thing as above, except that we require the program to produce a point in $\mathbb{Z}^n$, not just $\mathbb{R}^n$. Although linear programming and convex programming are known to be in **P**, integer programming is a well-known NP-complete problem.

4.2.2 Lenstra's algorithm

Theorem 3 (Lenstra) *If our polytope/convex body is in $\mathbb{R}^n$ for any constant n, then there exists a polynomial time algorithm for integer programming.*

Remark.

- For linear programming (LP), the running time of the algorithm will grow exponentially in n, but polynomially in m (the number of constrains) and the number of bits in the inputs.
- For convex programming, the running time is polynomial in $\log(R/r)$.
- As before, we could also ask for maximum of $c \cdot x$ over all $x \in K \cap Z^n$, which is equivalent to the feasibility problem, as we can do a binary search on the whole range of $c \cdot x$.

The main idea of Lenstra's algorithm is the following. The main difficulty of integer programming comes from the fact that K may not be well-rounded, therefore it could be exponentially large but still contain no integral point, as illustrated in the following figure:

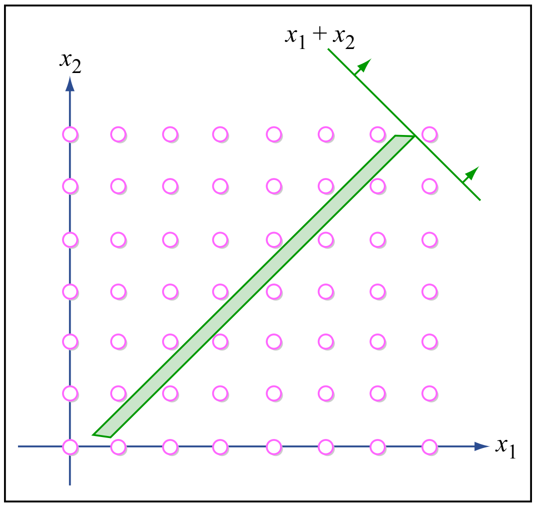

Figure by MIT OpenCourseWare.

Figure 1: A not-well-rounded convex body

Our first step is thus to change the basis so that K is well-rounded, i.e., K contains a ball of radius 1 and is contained in a ball of radius $c(n)$ for some function that depends only on n. Such a transformation will sends $\mathbb{Z}^n$ to some lattice L. Now our convex body is well-rounded but the basis of lattice L may be ill-conditioned, as shown in the following figure:

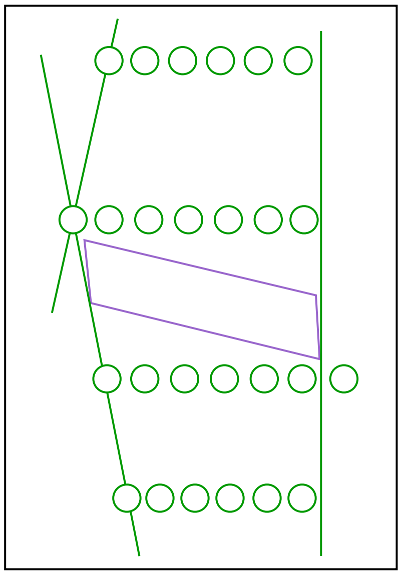

Figure by MIT OpenCourseWare.

Figure 2: A well-rounded convex body and an ill-conditioned lattice basis

It turns out that the lattice points are still well-separated and we can remedy the lattice basis by a basis reduction procedure of LLL (i.e., discrete Gram-Schmidt). Finally we chop the lattice space up in some intelligent way and search for lattice points in K.

Note that in the first step of Lenstra's algorithm, what we need is an algorithmic version of Fritz John's theorem. As we saw in the problem set, there is an efficient algorithm which, for any convex body K specified by a separation oracle, constructs an ellipsoid E such that

$$E(P') \subseteq K \subseteq O(n^{3/2})E(P').$$

Next let $T : \mathbb{R}^n \to \mathbb{R}^n$ be the linear transformation such that $E(P')$ is transformed to $\mathbf{B}(P, 1)$. Now K is sandwiched between two reasonably-sized balls:

$$\mathbf{B}(P, 1) \subseteq TK \subseteq \mathbf{B}(P, R),$$

where $R = O(n^{3/2})$ is the radius of the outer ball.

Let $L = T\mathbb{Z}^n$ with basis $Te_1, \ldots, Te_n$. Our goal is to find a point (if it exists) in $TK \cap T\mathbb{Z}^n = TK \cap L$. Our next step is to apply the basis reduction in LLL algorithm. We will need the following two lemmas in analyzing Lenstra's algorithm. The proofs of the lemmas are left as exercises.

Lemma 4 *Let $b_1, \ldots, b_n$ be any basis for L with $||b_1||^2 \leq \cdots \leq ||b_n||^2$. Then for every $x \in \mathbb{R}^n$, there exists a lattice point y such that*

$$\begin{aligned} ||x - y||^2 &\leq \frac{1}{4}(||b_1||^2 + \cdots + ||b_n||^2) \\ &\leq \frac{1}{4}n||b_n||^2. \end{aligned}$$

Lemma 5 *For a reduced basis $b_1, \ldots, b_n$ ordered as above,*

$$\prod_{i=1}^{n} ||b_i|| \leq 2^{n(n-1)/4} det(L).$$

Consequently, if we let $H = span(b_1, \ldots, b_{n-1})$, then

$$2^{-n(n-1)/4}||b_n|| \leq dist(H, b_n) \leq ||b_n||.$$

Let $b_1, \ldots, b_n$ be a reduced basis for L. Applying Lemma 4 gives us a point $y \in L$ such that $||y - P|| \leq \frac{1}{2}\sqrt{n}||b_n||$.

- case 1: $y \in TK$. We find a point in $TK \cap L$.
- case 2: $y \notin TK$, hence $y \notin \mathbf{B}(P, 1)$. Consequently, $||y - P|| \geq 1$ and $||b_n|| \geq \frac{2}{\sqrt{n}}$.

This means that the length of b_n is not much smaller than R. In the following we partition L along the sublattice "orthogonal" to b_n and then apply this process recursively.

Let L' be the lattice spanned by $b_1, \ldots, b_{n-1}$ and let $\mathcal{L}_i = L' + ib_n$ for each $i \in \mathbb{Z}$. Clearly $L = \bigcup_{i \in \mathbb{Z}} \mathcal{L}_i$. From Lemma 5 the distance between two adjacent hyperplanes is at least

$$\begin{aligned} \text{dist}(b_n, \text{span}(b_1, \ldots, b_{n-1})) &\geq 2^{-n(n-1)/4}||b_n|| \\ &\geq \frac{2}{\sqrt{n}}2^{-n(n-1)/4}||b_n|| = c_1(n), \end{aligned}$$

where $c_1(n)$ is some function that depends only on n. This implies that the convex body TK can not intersect with too many hyperplanes. That is

$$|\{i \in \mathbb{Z} : \mathcal{L}_i \cap \mathbf{B}(P, R) \neq \emptyset\}| \leq 2R/c_1(n) = c_2(n)$$

for some function $c_2(n)$ that depends only on n. Now we have reduced our original searching problem in n-dimensional space to $c_2(n)$ instances of searching problems in $(n-1)$-dimensional space. Therefore we can apply this process recursively and the total running time will be a polynomial in the input size times a function that depends only on n.

LLL lattice basis reduction algorithm

Helfer Etienne

21.03.2010

Contents

1 Lattice

1.1 Introduction

Since the LLL lattice reduction basis algorithm operates on a lattice it is important to understand what is it. Many concepts in Lattice theory are related with linear algebra : a lattice can be represented with the matrix of its basis, it has a determinant, and so on. Later we will need linear algebra methods and matrix properties for the LLL algorithm. I won't give a complete and precise view of the lattice theory but favor the geometrical point of view and focus on the elements that are needed to understand LLL basis reduction.

1.2 Definition

A lattice is a discrete subgroup of an Euclidean vector space. In general the vector space is $\mathbb{R}^n$ or a subspace of $\mathbb{R}^n$. It is conveniant to describe a lattice using its basis. The basis of a lattice is a set of linearly independent vectors in $\mathbb{R}^n$ which can generate the lattice by combining them. Notice that different bases can generate the same lattice (cf. figure 1).

Definition 1. *A set of vectors* $\{\mathbf{b_1}, \mathbf{b_2}, \dots, \mathbf{b_m}\}$ *in* $\mathbb{R}^n$ *is linearly independent if the equation*

$$c_1\mathbf{b_1} + c_2\mathbf{b_2} + \cdots + c_m\mathbf{b_m} = \mathbf{0} \text{ where } c_i \in \mathbb{R} \tag{1}$$

accepts only the trivial solution $c_1 = c_2 = \cdots = c_m = 0$

Theorem 1. *If a set of vectors in* $\mathbb{R}^n$ *contains more vectors than n (if* $m > n$*), then this set of vectors is not linearly independent.*

Definition 2. *A subspace of* $\mathbb{R}^n$ *is a an arbitrary set H that has the following properties :*

1. *the nul vector* $\mathbf{0}$ *is an element of H*
2. *H is close under addition : for every* $\mathbf{u}$ *and* $\mathbf{v}$ *in H, their sum* $\mathbf{u} + \mathbf{v}$ *is an element of H*
3. *H is close under scalar multiplication : for every* $\mathbf{u}$ *in H and scalar c, the vector* $c\mathbf{u}$ *is an element of H*

Notice that $\mathbb{R}^n$ is a subspace of $\mathbb{R}^n$

Definition 3. *A basis B of a subspace H of* $\mathbb{R}^n$ *is a set of linearly independent vectors in* $\mathbb{R}^n$ *that generates H.*

$$B = \{\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_m}\} \text{ where } \mathbf{b_i} \in \mathbb{R}^n \tag{2}$$

$$H = \sum_{i=1}^{m} \mathbb{R}\mathbf{b_i} = \left\{ \sum_{i=1}^{m} c_i\mathbf{b_i} \text{ where } c_i \in \mathbb{R},\ \mathbf{b_i} \in \mathbb{R}^n \right\} \tag{3}$$

Definition 4. *A lattice* Λ *is a discrete subgroup of H generated by all the integer combinations of the vectors of some basis B :*

$$\Lambda = \sum_{i=1}^{m} \mathbb{Z}\mathbf{b_i} = \left\{ \sum_{i=1}^{m} z_i\mathbf{b_i} \text{ where } z_i \in \mathbb{Z},\ \mathbf{b_i} \in \mathbb{R}^n \right\} \tag{4}$$

Definition 5. *The rank m of a lattice Λ generated by a basis B of a subspace H of $\mathbb{R}^n$ is the number of vectors in B.*

Theorem 1 implies that $m \leq n$. If $m = n$ then H is $\mathbb{R}^n$ and Λ is a full rank lattice.

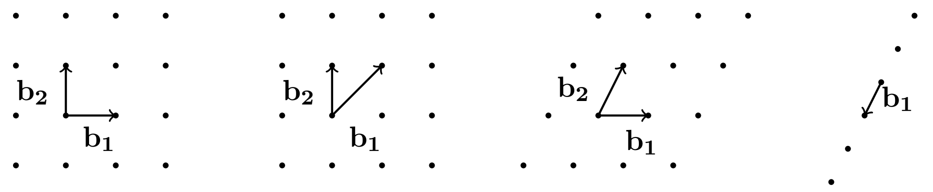

Figure 1: Examples of lattices generated with different bases in $\mathbb{R}^2$. The first and the second lattice are the same. The first three lattices are rank 2 and the fourth is rank 1.

1.3 Determinant

The determinant of a lattice Λ ($\det \Lambda$) is an important numerical invariant of Λ. Geometrically speaking, $\det \Lambda$ is the volume of the parallelepiped spanned by the basis. The determinant does not depend on the choice of the basis.

Another more general perspective is to consider $\det \Lambda$ as the inverse of the volume density of elements in Λ.

Definition 6. *The volume of a n-dimensional ball B of radius r is given by proposition*

$$\text{vol}B(r,n) = r^n \text{vol}B(1,n) = \frac{r^n \pi^{n/2}}{\frac{n}{2}!} \tag{5}$$

where $\frac{n}{2}!$ is inductively defined by $0! = 1$, $\frac{1}{2}! = \frac{\sqrt{\pi}}{2}$, and $\frac{n}{2}! = \frac{n}{2}\frac{n-2}{2}!$

Definition 7 (Determinant definition)**.** $\det \Lambda$ *is the volume of the m-dimensional ball B, where m is the rank of Λ, divided by the number of elements belonging to Λ in B when radius of B tends to infinity.*

$$\det \Lambda = \lim_{r \to \infty} \frac{\text{vol}B(r,m)}{\#\{y \in \Lambda \, where \, \|y\| \leq r\}} \tag{6}$$

Theorem 2. *Given a lattice Λ with a basis $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_i}\}$ then the determinant is equal to the volume of the parallelepiped spanned by $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}$.*

Proof. Argument : It is coherent with our definition because when the radius of the ball r is big then $volB(r,n) \approx \#\{y \in \Lambda \, where \, \|y\| \leq r\}$ times the volume of the parallelepiped. Figure 2 is an illustration of this approximation. $\square$

Notice that for a full rank lattice, from linear algebra we know that $|\det[\mathbf{b_1}\ \mathbf{b_2}\ \ldots\ \mathbf{b_n}]|$ is the volume of the parallelepiped spanned by the basis vectors $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}$, so

$$\det \Lambda = |\det[\mathbf{b_1}\ \mathbf{b_2}\ \ldots\ \mathbf{b_n}]|$$

Theorem 3 (Hadamard's inequality)**.** *The determinant is less than or equal to the product of the norm of the basis vectors.*

$$\det \Lambda \leq \prod_{i=1}^{m} \|\mathbf{b_i}\| \tag{7}$$

Equality holds if and only if the vectors $\mathbf{b_i}$ are pairwise orthogonal (if $\mathbf{b_i} \cdot \mathbf{b_j} = \mathbf{0}$ when $i \neq j$ and $\cdot$ is the scalar product).

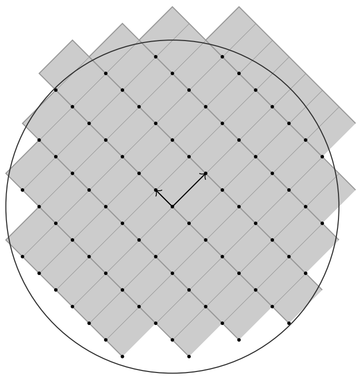

Figure 2: Illustration with a lattice of rank 2 that the volume of the ball approximates $\det \Lambda$ times the number of elements in Λ lying inside the ball.

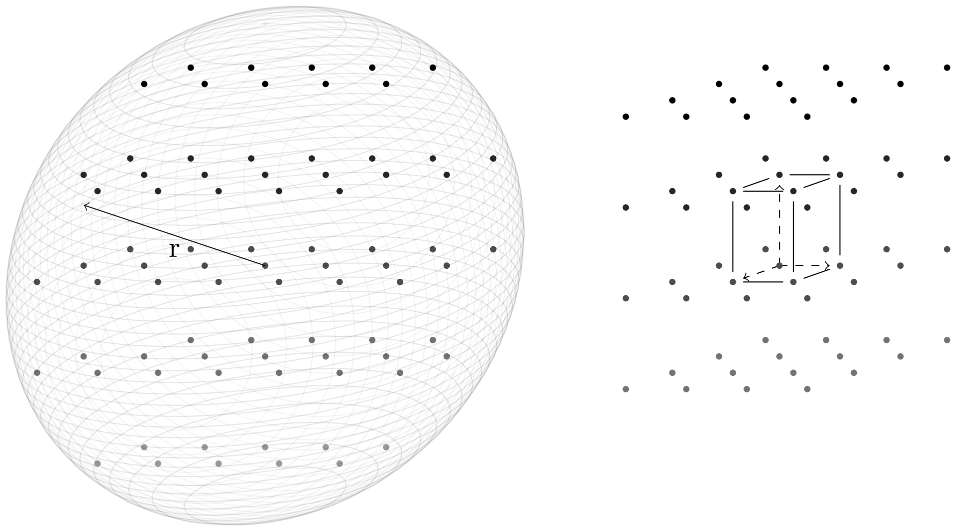

Figure 3: A lattice Λ of rank 3. On the left $\det \Lambda$ is represented as the ball and the elements of Λ inside. On the right $\det \Lambda$ is represented as the volume of the parallelepiped spanned by a basis of Λ.

1.4 Shortest vector problem

The shortest vector is the following : given a lattice Λ find a shortest vector $\mathbf{v}$ among the set of vectors going from the zero element to any non-zero element x in Λ. Notice that $-\mathbf{v}$ is also a shortest vector. In an algorithmic context, one may take 'shortest possible' to mean : shortest possible given the time one is willing to spend. The main theoretical result about this is the Minkowski's theorem which gives us an upper bound for the shortest vector.

Theorem 4 (Minkowski's theorem). *Given a lattice Λ of rank m, if λ is the norm of the shortest vector then :*

$$\lambda \leq \frac{2}{\sqrt{\pi}} \frac{m}{2}!^{\frac{1}{m}} \det \Lambda^{\frac{1}{m}} \tag{8}$$

Proof. Argument : If we place a m-dimensional ball of radius $\frac{\lambda}{2}$ on each element of Λ one can see that the balls are pairwise disjoint (cf Figure 4). From that one deduces that the volume of the ball is less than or equal than the determinant and the theorem follows :

$$volB(\frac{\lambda}{2}, m) \leq \det \Lambda$$

$$\frac{\frac{\lambda}{2}^m \pi^{m/2}}{\frac{m}{2}!} \leq \det \Lambda$$

$$\frac{\lambda}{2}^m \leq \frac{\frac{m}{2}! \det \Lambda}{\pi^{\frac{m}{2}}}$$

$$\lambda \leq \frac{2}{\sqrt{\pi}} \frac{m}{2}!^{\frac{1}{m}} \det \Lambda^{\frac{1}{m}}$$

Equality holds only with lattices of rank 1. □

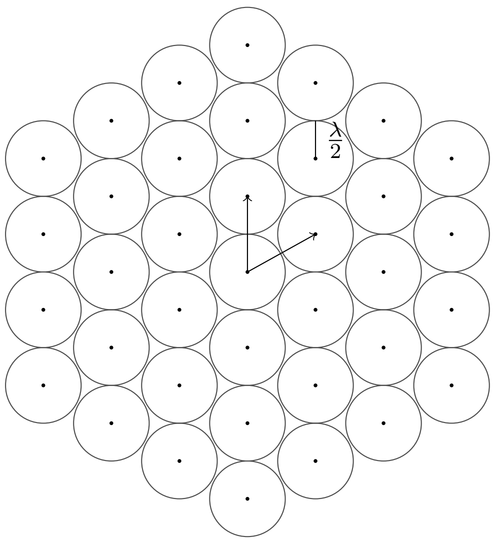

Figure 4: Illustration of a rank 2 lattice. One can see that the balls of radius $\frac{\lambda}{2}$ are pairwise disjoint. From that and the fact that the determinant is the inverse of the volume density of elements one concludes that $\det \Lambda < \frac{\pi\lambda^2}{4}$.

2 Basis reduction

2.1 Introduction

The idea of the basis reduction is to change a basis B of a lattice Λ into a shorter basis B' such that Λ remains the same. To do this we can use these following operations :

1. Swapping 2 vectors of the basis. As the swapping changes only the order of vectors in the basis it is trivial that Λ is not affected.

2. Replacing $\mathbf{b_j}$ by $-\mathbf{b_j}$. It is trivial that Λ is not affected.

3. Adding (or substracting) to a vector $\mathbf{b_j}$ a linear and discrete combination of the other vectors of the basis. The lattice is not affected because

if we take an arbitrary vector $\mathbf{v}$ which belongs to Λ we can express it as a discrete combination of the vectors of the basis : $\mathbf{v} = \sum_{i=1}^{m} z_i \mathbf{b_i}$ and if then we replace $\mathbf{b_j}$ by a discrete combination of the other vectors of the basis : $\mathbf{b_j} \leftarrow \mathbf{b_j} + \sum_{i \neq j} y_i \mathbf{b_i}$ we can still express $\mathbf{v}$ as a discrete combination of the vectors of the new basis : $\mathbf{v} = \sum_{i \neq j} z_i \mathbf{b_i} + z_j(\mathbf{b_j} - \sum_{i \neq j} y_i \mathbf{b_i})$. In a similar way we can show that if $\mathbf{v}$ belongs not to Λ, then we cannot express it with a discrete combination of the new basis. It follows that the 2 bases generate the exact same lattice.

Basis reduction can be used to solve the shortest vector problem in the sense that the shortest vector of the basis ($\mathbf{b_1}$ in the basis reduction algorithms we will see) is very short. In rank 2 for a reduced basis we have that $\mathbf{b_1}$ is the shortest vector of Λ and we can get it in polynomial time. But for higher ranks there is no known algorithms that finds the shortest vector in polynomial time. The LLL basis reduction algorithm finds a fairly short vector in polynomial time and it is often sufficient for applications.

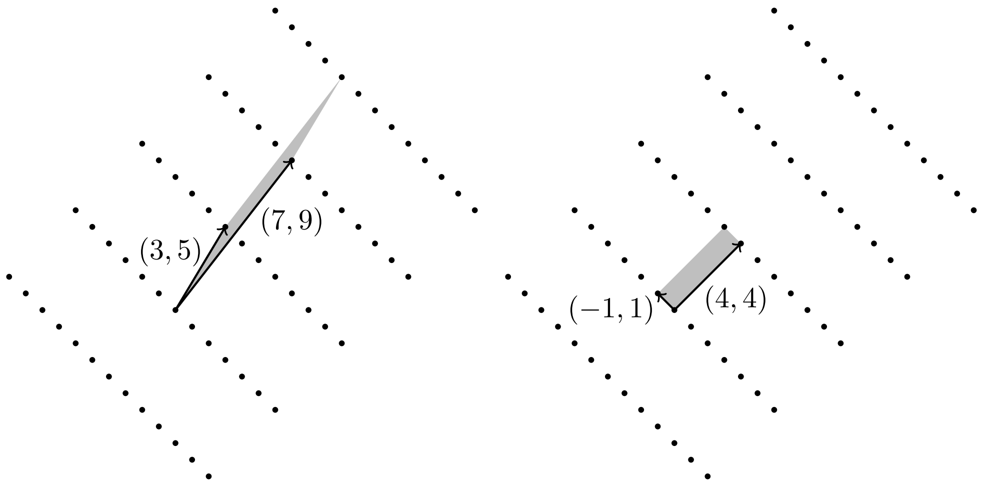

Figure 5: A lattice of rank 2 with two different bases. The determinant is depicted as the area of the parallelogram defined by the basis. The second basis is reduced and orthogonal.

2.2 Rank 2 basis reduction

Basis reduction of rank 2 lattices is easy to understand and plays a pivotal role in LLL basis reduction algorithm. We start with a basis $\{\mathbf{b_1}, \mathbf{b_2}\}$ and we try to reduce it. If $\mathbf{b_1}$ is shorter than $\mathbf{b_2}$ the intuitive approach is to substract from $\mathbf{b_2}$ an integer multiple z of $\mathbf{b_1}$. We want to choose z such that the new vector $\mathbf{b_2} - z\mathbf{b_1}$ is as short as possible. To solve this problem we take for z the coefficient u of the orthogonal projection of $\mathbf{b_2}$ on $\mathbf{b_1}$ (cf figure 6) rounded to the nearest integer. We repeat this process until we can no longer reduce the basis.

Definition 8 (Reduced basis in rank 2)**.** *A basis* $\{\mathbf{b_1}, \mathbf{b_2}\}$ *is said to be reduced if and only if the norm of* $\mathbf{b_1}$ *is less than or equal to the norm of* $\mathbf{b_2}$ *and the absolute value of the orthogonal projection coeffecient* $u = \frac{\mathbf{b_1}\cdot\mathbf{b_2}}{\mathbf{b_1}\cdot\mathbf{b_1}}$ *is less than or equal to* $\frac{1}{2}$*.*

$$\{\mathbf{b_1}, \mathbf{b_2}\}\,\textit{is reduced} \iff \|\mathbf{b_1}\| \le \|\mathbf{b_2}\|\,\textit{and}\,\frac{|\mathbf{b_1}\cdot\mathbf{b_2}|}{\mathbf{b_1}\cdot\mathbf{b_1}} \le \frac{1}{2} \tag{9}$$

To picture this observe that in figure 7 given an arbitrary $\mathbf{b_1}$ the basis is reduced if and only if $\mathbf{b_2}$ lies on the shaded area.

Theorem 5. *Given a lattice* Λ *of rank 2, if* λ *is the norm of the shortest vector then :*

$$\lambda \le \sqrt{\frac{2}{\sqrt{3}}\det\Lambda} \tag{10}$$

Proof. Suppose that we have a reduced basis $\{\mathbf{b_1}, \mathbf{b_2}\}$ of Λ. Using orthogonal projection and the properties of reduced bases we get :

$$\mathbf{b_2} = \mathbf{b_2^*} + u\mathbf{b_1}$$

$$\|\mathbf{b_2}\|^2 = \|\mathbf{b_2^*}\|^2 + u^2\|\mathbf{b_1}\|^2$$

$$\|\mathbf{b_2^*}\|^2 = \|\mathbf{b_2}\|^2 - u^2\|\mathbf{b_1}\|^2 \ge \|\mathbf{b_1}\|^2 - \frac{1}{4}\|\mathbf{b_1}\|^2 = \frac{3}{4}\|\mathbf{b_1}\|^2$$

$$\|\mathbf{b_2^*}\| \ge \frac{\sqrt{3}}{2}\|\mathbf{b_1}\|$$

$$\|\mathbf{b_2^*}\|\|\mathbf{b_1}\| = \det\Lambda \ge \frac{\sqrt{3}}{2}\|\mathbf{b_1}\|^2$$

$$\sqrt{\frac{2}{\sqrt{3}}\det\Lambda} \ge \|\mathbf{b_1}\|$$

It gives us for rank 2 lattice a new bound for λ which is better than the bound given by the Minkowski's theorem (cf theorem 4). □

Theorem 6. *If a basis* $\{\mathbf{b_1}, \mathbf{b_2}\}$ *of* Λ *is reduced then* $\mathbf{b_1}$ *is a shortest vector of* Λ*.*

$$\{\mathbf{b_1}, \mathbf{b_2}\}\,\textit{is reduced} \Rightarrow \mathbf{b_1}\,\textit{is a shortest vector.} \tag{11}$$

Proof. Let $\mathbf{x}$ be a shortest vector of $\Lambda - \{\mathbf{0}\}$. We can express it with the reduced basis : $\mathbf{x} = z_1\mathbf{b_1} + z_2\mathbf{b_2}$. We have $\|\mathbf{x}\|^2 = \|z_1\mathbf{b_1} + z_2(\mathbf{b_2^*} + u\mathbf{b_1})\|^2 = (z_1 - z_2u)^2\|\mathbf{b_1}\|^2 + z_2^2\|\mathbf{b_2^*}\| \ge (z_1 - z_2u)^2\|\mathbf{b_1}\|^2 + \frac{3}{4}z_2^2\|\mathbf{b_1}\|^2$.

$$\|\mathbf{x}\|^2 \ge (z_1 - z_2u)^2\|\mathbf{b_1}\|^2 + \frac{3}{4}z_2^2\|\mathbf{b_1}\|^2$$

1. for $z_2 = 0$ and $z_1 \neq 0$: $\|\mathbf{x}\|^2 \geq z_1^2\|\mathbf{b_1}\|^2 \geq \|\mathbf{b_1}\|^2$
2. for $|z_2| = 1$: $\|\mathbf{x}\|^2 \geq (z_1 \pm u)^2\|\mathbf{b_1}\|^2 + \frac{3}{4}\|\mathbf{b_1}\|^2 \geq u^2\|\mathbf{b_1}\|^2 + \frac{3}{4}\|\mathbf{b_1}\|^2 \geq \frac{1}{4}\|\mathbf{b_1}\|^2 + \frac{3}{4}\|\mathbf{b_1}\|^2 = \|\mathbf{b_1}\|^2$
3. for $|z_2| \geq 2$: $\|\mathbf{x}\|^2 \geq (z_1 - z_2u)^2\|\mathbf{b_1}\|^2 + \frac{3}{4}4\|\mathbf{b_1}\|^2 > \|\mathbf{b_1}\|^2$

So we have $\|\mathbf{x}\|^2 \geq \|\mathbf{b_1}\|^2$ and as $\mathbf{x}$ is a shortest vector only equality can hold : $\|\mathbf{x}\|^2 = \|\mathbf{b_1}\|^2$. We conclude that $\mathbf{b_1}$ is also a shortest vector. $\square$

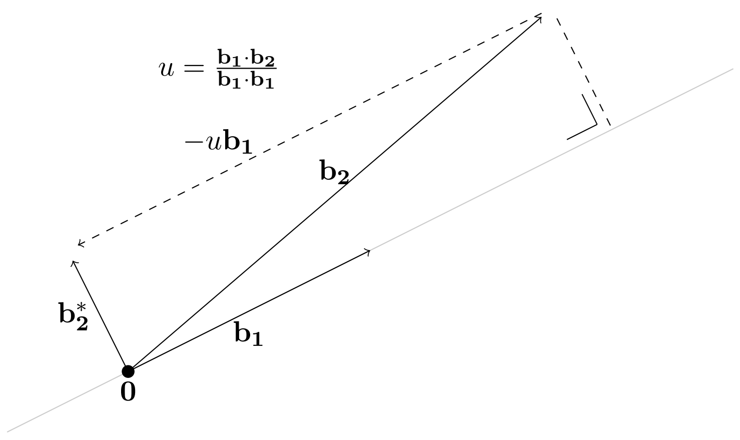
$u = \frac{\mathbf{b_1}\cdot\mathbf{b_2}}{\mathbf{b_1}\cdot\mathbf{b_1}}$
$-u\mathbf{b_1}$
$\mathbf{b_2}$
$\mathbf{b_2^*}$
$\mathbf{b_1}$
$\mathbf{0}$

Figure 6: Orthogonal projection. The vector $u\mathbf{b_1}$ is called the orthogonal projection of $\mathbf{b_2}$ on $\mathbf{b_1}$. The set $\{\mathbf{b_1}, \mathbf{b_2^*}\}$ is an orthogonal basis for the subspace generated by $\{\mathbf{b_1}, \mathbf{b_2}\}$. Notice that the lattice Λ generated by $\{\mathbf{b_1}, \mathbf{b_2}\}$ has $\det \Lambda = \|\mathbf{b_1}\|\|\mathbf{b_2^*}\|$.

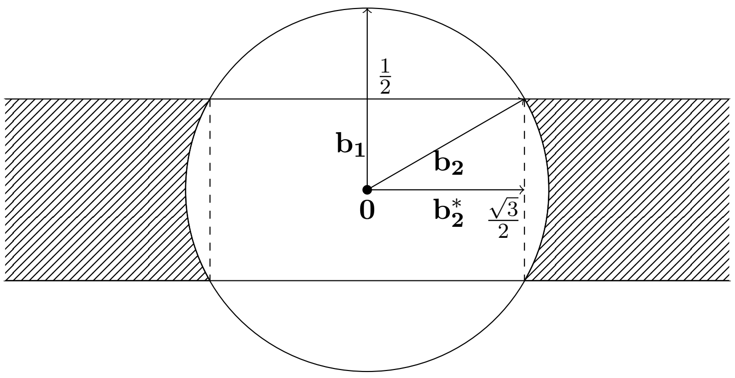

Figure 7: If the basis is reduced then $\mathbf{b_2}$ lies on the shaded area. We can see that for a reduced basis we have $\|\mathbf{b_2^*}\| \geq \frac{\sqrt{3}}{2}\|\mathbf{b_1}\|$.

Algorithm 1: Rank 2 basis reduction.

```
input  : basis { $\mathbf{b_1}$, $\mathbf{b_2}$ }
output: reduced basis { $\mathbf{b_1}$, $\mathbf{b_2}$ }
if $\|\mathbf{b_1}\| > \|\mathbf{b_2}\|$ then
    $swap(\mathbf{b_1}, \mathbf{b_2})$
while $\|\mathbf{b_1}\| < \|\mathbf{b_2}\|$ do
    $u = (\mathbf{b_1} \cdot \mathbf{b_2})/(\mathbf{b_1} \cdot \mathbf{b_1})$
    $\mathbf{b_2} = \mathbf{b_2} - round(u)\mathbf{b_1}$
    $swap(\mathbf{b_1}, \mathbf{b_2})$
$swap(\mathbf{b_1}, \mathbf{b_2})$ // to have $\|\mathbf{b_1}\| \leq \|\mathbf{b_2}\|$
```

2.3 LLL basis reduction

The LLL-reduction algorithm (Lenstra Lenstra Lovász lattice basis reduction) is a polynomial time lattice reduction algorithm invented by Arjen Lenstra, Hendrik Lenstra and László Lovász in 1982. Since no efficient (polynomial time) algorithm is known to solve the shortest vector problem exactly in arbitrary high dimension, LLL is used to get an approximation of the shortest vector. This approximation is sufficient for many applications.

Roughly speaking, LLL performs successives orthogonal projections, if necessary swapping 2 consecutives vectors of the basis, in order to get a reduced or near orthogonal basis.

Theorem 7. *Gram-Schmidt orthogonalization method. Given a basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ *of a subspace* H_m *of* $\mathbb{R}^n$*, we define :*

$$\mathbf{b_1^*} = \mathbf{b_1}$$

$$\mathbf{b_2^*} = \mathbf{b_2} - \frac{\mathbf{b_2} \cdot \mathbf{b_1^*}}{\mathbf{b_1^*} \cdot \mathbf{b_1^*}} \mathbf{b_1^*}$$

$$\mathbf{b_3^*} = \mathbf{b_3} - \frac{\mathbf{b_3} \cdot \mathbf{b_1^*}}{\mathbf{b_1^*} \cdot \mathbf{b_1^*}} \mathbf{b_1^*} - \frac{\mathbf{b_3} \cdot \mathbf{b_2^*}}{\mathbf{b_2^*} \cdot \mathbf{b_2^*}} \mathbf{b_2^*}$$

$$\vdots$$

$$\mathbf{b_m^*} = \mathbf{b_m} - \frac{\mathbf{b_m} \cdot \mathbf{b_1^*}}{\mathbf{b_1^*} \cdot \mathbf{b_1^*}} \mathbf{b_1^*} - \frac{\mathbf{b_m} \cdot \mathbf{b_2^*}}{\mathbf{b_2^*} \cdot \mathbf{b_2^*}} \mathbf{b_2^*} - \cdots - \frac{\mathbf{b_m} \cdot \mathbf{b_{m-1}^*}}{\mathbf{b_{m-1}^*} \cdot \mathbf{b_{m-1}^*}} \mathbf{b_{m-1}^*}$$

Then for $1 \leq k \leq m$ *and* H_k *the subspace generated by the basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_k}\}$

$$\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_k^*}\} \textit{ is an orthogonal basis of } H_k \tag{12}$$

Proof. 1. For $k = 1$: It is trivial because $\mathbf{b_1^*} = \mathbf{b_1}$.

2. For $k = 2$:

   - $\{\mathbf{b_1^*}, \mathbf{b_2^*}\}$ is orthogonal because $\mathbf{b_2^*}$ is constructed using the orthogonal projection of $\mathbf{b_2}$ on $\mathbf{b_1^*}$.
   - As $\mathbf{b_2^*}$ is obtained substracting from $\mathbf{b_2}$ a multiple of $\mathbf{b_1^*}$ or equally a multiple of $\mathbf{b_1}$ since $\mathbf{b_1^*} = \mathbf{b_1}$ and the fact that substracting from a vector of a basis a linear combination of the other vectors of the basis do not modify the subspace then it follows that $\{\mathbf{b_1^*}, \mathbf{b_2^*}\}$ is a basis of H_2.

3. For $2 < k \leq m$:

   - $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_k^*}\}$ is orthogonal because $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_{k-1}^*}\}$ is an orthogonal basis by induction hypothesis and $\mathbf{b_k^*}$ is constructed using successive orthogonal projections of $\mathbf{b_k^*}$ on the vectors $\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_{k-1}^*}$ such that $\mathbf{b_k^*}$ is pairwise orthogonal with them.
   - As $\mathbf{b_k^*}$ is obtained substracting from $\mathbf{b_k}$ a linear combination of the vectors $\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_{k-1}^*}$ or equally a linear combination of the vectors $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_{k-1}}$ since by induction hypothesis we have that $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_{k-1}^*}\}$ is a basis of H_{k-1} and the fact that substracting from a vector of a basis a linear combination of the other vectors of the basis do not modify the subspace then it follows that $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_k^*}\}$ is a basis of H_k.

□

QR matrix factorisation : If we define the matrices $B = [\mathbf{b_1}\ \mathbf{b_2}\ \ldots\ \mathbf{b_m}]$, $Q = [\mathbf{b_1^*}\ \mathbf{b_2^*}\ \ldots\ \mathbf{b_m^*}]$, and $R = [\mathbf{u_1}\ \mathbf{u_2}\ \ldots\ \mathbf{u_j}]$ such that $\mathbf{u_i} \in R^m$ and the j^{th} element of $\mathbf{u_i}$ is defined as follows :

- $\mathbf{u_i}[j] = (\mathbf{b_i} \cdot \mathbf{b_j^*}) / (\mathbf{b_j^*} \cdot \mathbf{b_j^*})$ if $j < i$

- $\mathbf{u_i}[j] = 1$ if $j = i$
- $\mathbf{u_i}[j] = 0$ if $j > i$

One has $B = QR$. Doing the matrix multiplication one notices that this is an equivalent way of expressing the Gram-Schmidt orthogonalization method. Since R is an upper triangular matrix with only 1's on the diagonal then $\det R = 1$.

Theorem 8. *Given a lattice Λ generated by the basis $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ and the orthogonal basis $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$ obtained using the Gram-Schmidt method, then :*

$$\det \Lambda = \prod_{i=1}^{m} \|\mathbf{b_i^*}\| \tag{13}$$

Proof. First notice that Λ and the lattice generated by $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$ are not the same. The proof come from the fact that one can transform the basis $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ into the orthogonal basis $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$ using only the operation that consists of substracting from a vector of the basis a linear combination of the other vectors of the basis, which do not modify the volume of the parallelepiped spanned by the basis. Such transformation is done by rewriting the Gram-Schmidt method as follows :

$$\mathbf{b_1} \leftarrow \mathbf{b_1}$$

$$\mathbf{b_2} \leftarrow \mathbf{b_2} - \frac{\mathbf{b_2} \cdot \mathbf{b_1}}{\mathbf{b_1} \cdot \mathbf{b_1}} \mathbf{b_1}$$

$$\mathbf{b_3} \leftarrow \mathbf{b_3} - \frac{\mathbf{b_3} \cdot \mathbf{b_1}}{\mathbf{b_1} \cdot \mathbf{b_1}} \mathbf{b_1} - \frac{\mathbf{b_3} \cdot \mathbf{b_2}}{\mathbf{b_2} \cdot \mathbf{b_2}} \mathbf{b_2}$$

$$\vdots$$

$$\mathbf{b_m} \leftarrow \mathbf{b_m} - \frac{\mathbf{b_m} \cdot \mathbf{b_1}}{\mathbf{b_1} \cdot \mathbf{b_1}} \mathbf{b_1} - \frac{\mathbf{b_m} \cdot \mathbf{b_2}}{\mathbf{b_2} \cdot \mathbf{b_2}} \mathbf{b_2} - \cdots - \frac{\mathbf{b_m} \cdot \mathbf{b_{m-1}}}{\mathbf{b_{m-1}} \cdot \mathbf{b_{m-1}}} \mathbf{b_{m-1}}$$

As $\det \Lambda$ is equal to the volume of the parallelepiped spanned by $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$, which is equal to the volume of the other parallelepiped spanned by $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$, which is a parallelepiped rectangle whose volume is equal to the product of its edges, one concludes that $\det \Lambda = \prod_{i=1}^{m} \|\mathbf{b_i^*}\|$.
Notice that if Λ is of a full rank lattice then using the QR factorisation we have that $B = QR$ which implies that $\det \Lambda = |\det B| = |\det Q||\det R| = |\det Q| = \prod_{i=1}^{m} \|\mathbf{b_i^*}\|$. $\square$

Definition 9. *c-Reduced basis. A basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ *is said to be c-reduced if and only if its orthogonal basis obtained with the Gram-Schmidt method* $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$ *verifies the following inequality for* $i = 1$ *to* $m - 1$ *:*

$$\|\mathbf{b_{i+1}^*}\|^2 \geq \frac{\|\mathbf{b_i^*}\|^2}{c} \tag{14}$$

A small value for c means a good reduction. Not every basis is 1-reducable, but each basis is $\frac{4}{3}$-reducable. The $\frac{4}{3}$ comes from the $\frac{\sqrt{3}}{2}$ one can see in figure 7. Figure 8 shows c-reduced basis in rank 2. Notice from Gram-Schmidt method that $\mathbf{b_1^*} = \mathbf{b_1}$.

Theorem 9. *Near-orthogonality of c-reduced basis. Given a lattice* Λ *and its c-reduced basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ *with* $c \geq \frac{4}{3}$ *then it is near-orthogonal in the sense that :*

$$\prod_{i=1}^{m} \|\mathbf{b_i}\| \leq c^{(m(m-1))/4} \det \Lambda \tag{15}$$

Proof. Multiplying for $i = 1$ to m the following inequality

$$\|\mathbf{b_i}\|^2 \leq c^{i-1} \|\mathbf{b_i^*}\|^2$$

and then taking the square root and using $\det \Lambda = \prod_{i=1}^{m} \mathbf{b_i^*}$ we conclude

$$\prod_{i=1}^{m} \|\mathbf{b_i}\| \leq c^{(m(m-1))/4} \det \Lambda$$

□

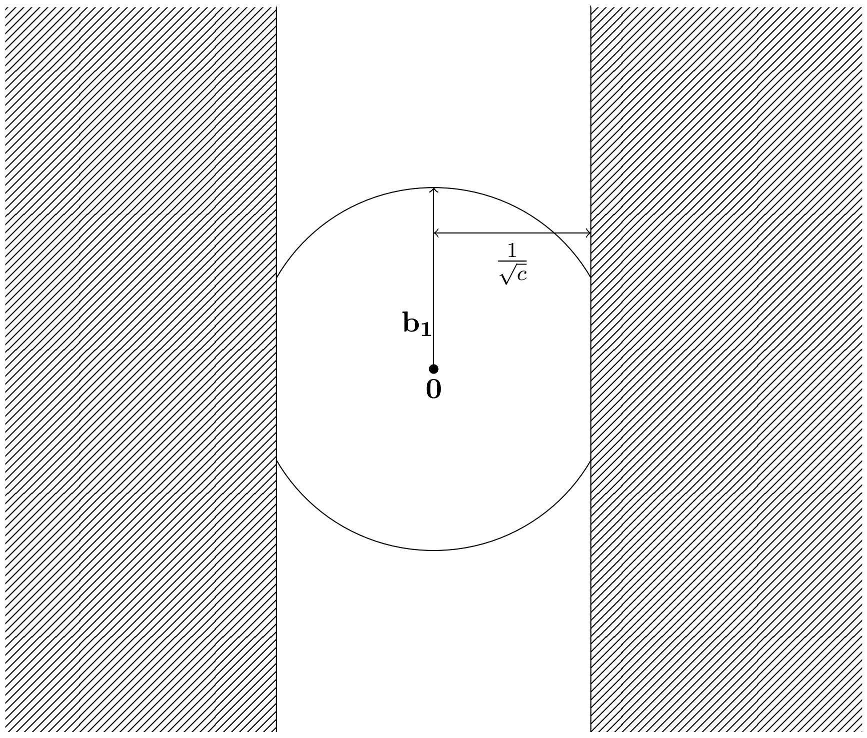

Figure 8: c-reduced basis in rank 2. Given an arbitrary $\mathbf{b_1}$, the basis is c-reduced if and only if $\mathbf{b_2}$ lies on the shaded area. Here $c = \frac{4}{3}$ which is the minimal value for c.

Theorem 10. *Shortest vector approximation with a c-reduced basis. If the basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ *is c-reduced and* λ *is the shortest vector norm then :*

$$\|\mathbf{b_1}\| \leq c^{(m-1)/4} \det \Lambda^{\frac{1}{m}} \tag{16}$$

$$\|\mathbf{b_1}\| \leq c^{(m-1)/2} \lambda \tag{17}$$

Proof. 1. From the definition of a c-reduced basis we have that for $i = 1$ to m

$$\|\mathbf{b_1}\|^2 = \|\mathbf{b_1^*}\|^2 \leq c^{i-1} \|\mathbf{b_i^*}\|^2.$$

Multiplying together all this inequalities we get

$$\|\mathbf{b_1}\|^{2m} \leq c^{0+1+2+\cdots+(m-1)} \prod_{i=1}^{m} \|\mathbf{b_i^*}\|^2.$$

As from theorem 8 we have $\det \Lambda = \prod_{i=1}^{m} \|\mathbf{b_i^*}\|$ and that $\sum_{i=0}^{m-1} i = \frac{m(m-1)}{2}$ we get

$$\|\mathbf{b_1}\|^{2m} \leq c^{m(m-1)/2} \det \Lambda^2.$$

Passing the inequality to the power $\frac{1}{2m}$ we get

$$\|\mathbf{b_1}\| \leq c^{(m-1)/4} \det \Lambda^{\frac{1}{m}}.$$

2. Let $x \in \Lambda - \mathbf{0}$, and let i be minimal such that $x \in \Lambda_i$ which is the sublattice of Λ generated by $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_i}\}$. Notice that $\|x\|$ is at least $\|\mathbf{b_i^*}\|$ then it follows that

$$\lambda^2 \geq \|x\|^2 \geq \|\mathbf{b_i^*}\|^2 \geq \frac{\|\mathbf{b_1^*}\|^2}{c^{(i-1)}} \geq \frac{\|\mathbf{b_1}\|^2}{c^{(m-1)}}.$$

Multiplying by c^{m-1} and passing to the power $\frac{1}{2}$ we prove that

$$c^{(m-1)/2}\lambda \geq \|\mathbf{b_1}\|.$$

□

Theorem 11. *For $c > \frac{4}{3}$ LLL finds a c-reduced basis in polynomial time*

Proof. We define the size of the orthogonal basis $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_m^*}\}$ as

$$s = \|\mathbf{b_1^*}\|^m \; \|\mathbf{b_2^*}\|^{m-1} \; \ldots \; \|\mathbf{b_m^*}\|$$

and $s_{initial}$ as the size of the basis orthogonal at the initialization of LLL and s_{final} the size of the orthogonal basis at termination of LLL. It will be usefull to analyse the effect of swapping $\mathbf{b_i^*}$ and $\mathbf{b_{i+1}^*}$ when $\|\mathbf{b_i^*}\|^2 > c\|\mathbf{b_{i+1}^*}\|^2$ on the new orthogonal basis $\{\mathbf{a_1^*}, \mathbf{a_2^*}, \ldots, \mathbf{a_m^*}\}$ and especially on s. Let s_b be the size before and s_a the size after the swapping and compare them :

1.
$$s_b = \|\mathbf{b_1^*}\|^m \; \|\mathbf{b_2^*}\|^{m-1} \; \ldots \; \|\mathbf{b_i^*}\|^{m-i+1} \; \|\mathbf{b_{i-1}^*}\|^{m-i} \; \ldots \; \|\mathbf{b_m^*}\|$$
$$s_b = \|\mathbf{b_1^*}\|^m \; \|\mathbf{b_2^*}\|^{m-1} \; \ldots \; \|\mathbf{b_i^*}\| \; (\|\mathbf{b_i^*}\| \; \|\mathbf{b_{i+1}^*}\|)^{m-i} \; \ldots \; \|\mathbf{b_m^*}\|$$

2.
$$s_a = \|\mathbf{a_1^*}\|^m \; \|\mathbf{a_2^*}\|^{m-1} \; \ldots \; \|\mathbf{a_i^*}\|^{m-i+1} \; \|\mathbf{a_{i+1}^*}\|^{m-i} \; \ldots \; \|\mathbf{a_m^*}\|$$

As we can notice it in the algorithm the swapping only affects the $\mathbf{b_i^*}$ and $\mathbf{b_{i+1}^*}$ (for the vectors before the i^{th} it is trivial and for the ones after the $(i+1)^{th}$ it comes from the fact that $\mathbf{b_{i+2}^*}$ is constructed using the orthogonal projection of $\mathbf{b_{i+2}}$ on the subspace $\{\mathbf{b_1^*}, \mathbf{b_2^*}, \ldots, \mathbf{b_{i+1}^*}\}$ which is not affected by the swapping and so on for $\mathbf{b_{i+3}^*}, \ldots, \mathbf{b_m^*}$) and we get :

$$s_a = \|\mathbf{b_1^*}\|^m \; \|\mathbf{b_2^*}\|^{m-1} \; \ldots \; \|\mathbf{a_i^*}\| \; (\|\mathbf{a_i^*}\| \; \|\mathbf{a_{i+1}^*}\|)^{m-i} \; \ldots \; \|\mathbf{b_m^*}\|$$

Algorithm 2: LLL basis reduction.

```
Input: basis { $\mathbf{b_1}$, $\mathbf{b_2}$, …, $\mathbf{b_m}$ } and $c$ such that $\mathbf{b_i} \in \mathbb{R}^n$ and $c \geq \frac{4}{3}$
Data: orthogonal basis { $\mathbf{b_1^*}$, $\mathbf{b_2^*}$, …, $\mathbf{b_m^*}$ } and orthogonal
      projection coefficent vectors { $\mathbf{u_1}$, $\mathbf{u_2}$, …, $\mathbf{u_m}$ } such that
      $\mathbf{b_i^*} \in \mathbb{R}^n$ and $\mathbf{u_i} \in \mathbb{R}^m$
Output: c-reduced basis { $\mathbf{b_1}$, $\mathbf{b_2}$, …, $\mathbf{b_m}$}

for $i \leftarrow 1$ to $m$ do // initialization
    $\mathbf{u_i} = \mathbf{0}$
    $\mathbf{u_i}[i] = 1$
    $\mathbf{b_i^*} = \mathbf{b_i}$
    for $j \leftarrow 1$ to $i - 1$ do
        $\mathbf{u_i}[j] = (\mathbf{b_i} \cdot \mathbf{b_j^*})/(\mathbf{b_j^*} \cdot \mathbf{b_j^*})$
        $\mathbf{b_i^*} = \mathbf{b_i^*} - \mathbf{u_i}[j]\,\mathbf{b_j^*}$
    reduce($i$)

while $i \leftarrow 1 < m$ do
    if $\|\mathbf{b_i^*}\|^2 \leq c\|\mathbf{b_{i+1}^*}\|^2$ then // $\{\mathbf{b_1}, \ldots, \mathbf{b_{i+1}}\}$ is c-reduced
        i = i + 1
    else
        /* modify Q and R in order to keep the relation B =
        QR after the swapping                    */
        $\mathbf{b_{i+1}^*} = \mathbf{b_{i+1}^*} + \mathbf{u_{i+1}}[i]\,\mathbf{b_i^*}$
        $\mathbf{u_i}[i] = (\mathbf{b_i} \cdot \mathbf{b_{i+1}^*})/(\mathbf{b_{i+1}^*} \cdot \mathbf{b_{i+1}^*})$
        $\mathbf{u_i}[i+1] = 1$
        $\mathbf{u_{i+1}}[i] = 1$
        $\mathbf{u_{i+1}}[i+1] = 0$
        $\mathbf{b_i^*} = \mathbf{b_i^*} - \mathbf{u_i}[i]\,\mathbf{b_{i+1}^*}$
        swap($\mathbf{u_i}, \mathbf{u_{i+1}}$)
        swap($\mathbf{b_i^*}, \mathbf{b_{i+1}^*}$)
        swap($\mathbf{b_i}, \mathbf{b_{i+1}}$)
        for $k \leftarrow i + 2$ to $m$ do
            $\mathbf{u_k}[i] = (\mathbf{b_k} \cdot \mathbf{b_i^*})/(\mathbf{b_i^*} \cdot \mathbf{b_i^*})$
            $\mathbf{u_k}[i+1] = (\mathbf{b_k} \cdot \mathbf{b_{i+1}^*})/(\mathbf{b_{i+1}^*} \cdot \mathbf{b_{i+1}^*})$
        if $|\mathbf{u_{i+1}}[i]| > \frac{1}{2}$ then reduce($i + 1$)
        $i = \max(i - 1, 1)$

Subroutine: reduce
  Input: $i$ such that $(i \leq m)$

  while $j \leftarrow (i - 1) > 0$ do
      $\mathbf{b_i} = \mathbf{b_i} - \text{round}(\mathbf{u_i}[j])\mathbf{b_j}$
      $\mathbf{u_i} = \mathbf{u_i} - \text{round}(\mathbf{u_i}[j])\mathbf{u_j}$
      $j = j - 1$
```

From theorem 8 we have $\det \Lambda = \prod_{k=1}^{m} \|\mathbf{b_k^*}\| = \prod_{i=k}^{k} \|\mathbf{a_k^*}\|$ and it implies that $\|\mathbf{b_i^*}\| \, \|\mathbf{b_{i+1}^*}\| = \|\mathbf{a_i^*}\| \, \|\mathbf{a_{i+1}^*}\|$ and we get :

$$s_a = \|\mathbf{b_1^*}\|^m \, \|\mathbf{b_2^*}\|^{m-1} \ldots \|\mathbf{a_i^*}\| \, (\|\mathbf{b_i^*}\| \, \|\mathbf{b_{i+1}^*}\|)^{m-i} \ldots \|\mathbf{b_m^*}\|$$

As we can notice it from the algorithm $\mathbf{a_i^*} = \mathbf{b_{i+1}^*} + \mathbf{u_{i+1}}\,[i]\,\mathbf{b_i^*}$ which is simply $\mathbf{b_{i+1}^*}$ without the projection component on $\mathbf{b_i^*}$. So $\|\mathbf{a_i^*}\|^2 = \|\mathbf{b_{i+1}^*}\|^2 + \mathbf{u_{i+1}}\,[i]^2\,\|\mathbf{b_i^*}\|^2$ and we get :

$$s_a = \|\mathbf{b_1^*}\|^m \|\mathbf{b_2^*}\|^{m-1} \ldots (\|\mathbf{b_{i+1}^*}\|^2 + \mathbf{u_{i+1}}\,[i]^2\,\|\mathbf{b_i^*}\|^2)^{\frac{1}{2}} (\|\mathbf{b_i^*}\| \|\mathbf{b_{i+1}^*}\|)^{m-i} \ldots \|\mathbf{b_m^*}\|$$

Combining together s_b and s_a we get :

$$s_b = \frac{s_a\,(\|\mathbf{b_{i+1}^*}\|^2 + \mathbf{u_{i+1}}\,[i]^2\,\|\mathbf{b_i^*}\|^2)^{\frac{1}{2}}}{\|\mathbf{b_i^*}\|}$$

Finally using $\mathbf{u_{i+1}}\,[i]^2 \leq \frac{1}{4}$ and multiplying by $\|\mathbf{b_i^*}\|$ and as $\|\mathbf{b_i^*}\|^2 > c\|\mathbf{b_{i+1}^*}\|^2$ we conclude :

$$s_b \leq \left(\frac{\|\mathbf{b_{i+1}^*}\|^2}{\|\mathbf{b_i^*}\|^2} + \frac{1}{4} \right)^{\frac{1}{2}} s_a < \left(\frac{1}{c} + \frac{1}{4} \right)^{\frac{1}{2}} s_a$$

From this result we observe that the number of times swapping happens in LLL is at most

$$\frac{2 \log (s_{initial}/s_{final})}{|\log (\frac{1}{c} + \frac{1}{4})|}$$

Notice that this expression is not defined for $c = \frac{4}{3}$ because a division by 0 occurs. So it suffices to take $c > \frac{4}{3}$ and a good lower bound for s_{final} to prove that LLL runs in polynomial time. □

3 LLL basis reduction for solving RSA problem

In this section, we will show how we can use Coppersmith's algorithm for finding small roots of univariate modular polynomials, which uses LLL, in order to attack the RSA cryptosystem under certain conditions.

3.1 RSA

Rivest Shamir Adleman or RSA is a very well known asymetric public key cryptographic algorithm used to exchange confidential information over the Internet. Here is how it works :

- Keys generation :

    1. Choose 2 prime numbers p and q.

2. Denote n as the product of p and q : $n = pq$.
3. Calculate Euler's totient function of n : $\phi(n) = (p-1)(q-1)$.
4. Choose an integer e coprime with $\phi(n)$: $\gcd(e, \phi(n)) = 1$.
5. Compute d as the inverse modulo $\phi(n)$ of e : $ed \equiv 1 \mod \phi(n)$.

- Encryption : $C = M^e \mod n$ where M is the message and C the message encrypted.
- Decryption : $M = C^d \mod n$ where M is the message and C the message encrypted.

Definition 10 (RSA problem)**.** *Given $M^e \mod N$ find $M \in \mathbb{Z}_N$.*

Definition 11 (Relaxed RSA problem : Small e, High Bits Known)**.** *Given $M^e, \tilde{M}$ with $|M - \tilde{M}| \leq N^{\frac{1}{e}}$ find $M \in \mathbb{Z}_N$.*

3.2 Coppersmith

In this section we present the Coppersmith method to find small roots of a monic univariate polynomial : We want to efficiently find all the solutions x_0 satisfying

$$f(x_0) = 0 \mod N \textit{ with } |x_0| \leq X \tag{18}$$

Theorem 12 (Howgrave-Graham)**.** *Let $g(x)$ be an univariate polynomial of degree δ. Further, let m be a positive integer. Suppose that*

$$g(x_0) = 0 \mod N^m \textit{ where } |x_0| \leq X \tag{19}$$

$$\|g(xX)\| < \frac{N^m}{\sqrt{\delta + 1}} \tag{20}$$

Then $g(x_0) = 0$ holds over the integers.

Proof. $|g(x_0)| = |\sum_{i=0}^{\delta} c_i x_0^i| \leq \sum_{i=0}^{\delta} |c_i x_0^i| \leq \sum_{i=0}^{n} |c_i| X^i \leq \sqrt{\delta + 1} \|g(xX)\| < N^m$. But $g(x_0)$ is a multiple of N^m and therefore it must be zero. □

Given the Howgrave-Graham theorem, the idea is to construct a collection $f_1(x), ..., f_n(x)$ of polynomials that all have the desired roots x_0 modulo N^m. Notice that for every integer linear combination g we have

$$g(x_0) = \sum_{i=1}^{n} a_i f_i(x_0) = 0 \mod N^m \text{ where } a_i \in \mathbb{Z}. \tag{21}$$

Then, using LLL basis reduction on the coefficient vectors of $f_i(xX)$, we might find a small coefficient vector $\mathbf{v}$ such that $\mathbf{v}$ respects the second condition of the Howgrave-Graham theorem : $\|\mathbf{v}\| < \frac{N^m}{\sqrt{n}}$ where n is the dimension of the coefficient vector $\mathbf{v}$.

Theorem 13 (Coppersmith)**.** *Let f(x) be a univariate monic polynomial of degree δ. Let N be an integer of unknown factorization. And let $\epsilon > 0$. Then we can find all solutions x_0 for the equation*

$$f(x) = 0 \mod N \textit{ with } |x_0| \leq \frac{1}{2} N^{\frac{1}{\delta}-\epsilon}. \tag{22}$$

Proof. To prove this we apply the coppersmith method. First we set $m = \lceil 1/(\delta\epsilon) \rceil$ and $X = \frac{1}{2}N^{\frac{1}{\delta}-\epsilon}$. Then we construct a collection of polynomials, where each polynomial has a root x_0 modulo N^m whenever $f(x)$ has the root x_0 modulo N. Here is the collection of polynomials we choose :

$$\begin{array}{lllll}
N^m & xN^m & x^2N^m & \ldots & x^{\delta-1}N^m \\
N^{m-1}f & xN^{m-1}f & x^2N^{m-1}f & \ldots & x^{\delta-1}N^{m-1}f \\
N^{m-2}f^2 & xN^{m-2}f^2 & x^2N^{m-2}f^2 & \ldots & x^{\delta-1}N^{m-2}f^2 \\
\vdots & \vdots & \vdots & & \vdots \\
Nf^{m-1} & xNf^{m-1} & x^2Nf^{m-1} & \ldots & x^{\delta-1}Nf^{m-1}
\end{array}$$

Or more compactly :

$$g_{i,j}(x) = x^j N^i f^{m-i}(x) \; for \; i = 1, \ldots, m \; and \; j = 0, \ldots, \delta - 1$$

We can see that $g_{i,j}(x_0) = 0 \mod N^m$ if $f(x_0) = 0 \mod N$ noticing that N^i is divisible i times by N and $f(x_0)^{m-i}$ is divisible $m-i$ times by N. Then we construct the lattice Λ that is spanned by the coefficent vectors of $g_{i,j}(xX)$:

$$\begin{bmatrix}
0 & 0 & & 0 & & & 0 & 0 & & NX^{\delta m-1} \\
\vdots & \vdots & & \vdots & & & \vdots & \vdots & \iddots & - \\
\vdots & \vdots & & \vdots & & & 0 & NX^{\delta m-\delta+1} & & \vdots \\
\vdots & \vdots & & \vdots & & & NX^{\delta m-\delta} & - & & \vdots \\
\vdots & \vdots & & \vdots & & \iddots & - & \vdots & & \vdots \\
\vdots & \vdots & & 0 & \iddots & \iddots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & N^mX^{\delta-1} & \iddots & \iddots & \vdots & \vdots & & \vdots \\
\vdots & 0 & \iddots & 0 & \iddots & \iddots & \vdots & \vdots & & \vdots \\
0 & N^mX & & \vdots & \iddots & \iddots & \vdots & \vdots & & \vdots \\
N^m & 0 & & 0 & \iddots & \iddots & - & - & & -
\end{bmatrix}$$

Notice that the rank of the lattice is δm. A nice thing about the matrix is that it is triangular. So we can easily compute the determinant of the lattice multiplying the terms on the diagonal :

$$\det \Lambda = N^{\frac{1}{2}\delta m(m+1)} X^{\frac{1}{2}\delta m(\delta m-1)}.$$

Then we reduce the lattice with LLL choosing $c = 2$ and we obtain a 2-reduced basis. We want the first vector of the basis $\mathbf{b_1}$ to satisfy the condition

$$\|\mathbf{b_1}\| < \frac{N^m}{\sqrt{\delta m}}$$

in order to apply the Howgrave-Graham theorem. Let's see if it does : After the LLL basis reduction we are guaranteed to have

$$\|\mathbf{b_1}\| \leq 2^{\frac{\delta m-1}{4}} \det \Lambda^{\frac{1}{\delta m}}.$$

So we need to prove that

$$2^{\frac{n-1}{4}} \det \Lambda^{\frac{1}{\delta m}} \leq \frac{N^m}{\sqrt{\delta m}}.$$

Using the fact that $\det \Lambda = N^{\frac{1}{2}\delta m(m+1)} X^{\frac{1}{2}\delta m(\delta m-1)}$, we obtain the new condition :

$$N^{\frac{\delta m(m+1)}{2\delta m}} X^{\frac{\delta m-1}{2}} \leq 2^{\frac{-(\delta m-1)}{4}} (\delta m)^{\frac{-1}{2}} N^m.$$

This gives us a condition on the size of X :

$$X \leq 2^{\frac{-1}{2}} (\delta m)^{\frac{-1}{\delta m-1}} N^{\frac{2m}{\delta m-1} - \frac{\delta m(m+1)}{\delta m(\delta m-1)}}.$$

Notice that $(\delta m)^{\frac{-1}{\delta m-1}} = 2^{\frac{-\log(\delta m)}{\delta m-1}} \geq 2^{\frac{-1}{2}}$ for $n > 6$. Therefore, our condition simplifies to

$$X \leq \frac{1}{2} N^{\frac{2m}{\delta m-1} - \frac{m+1}{\delta m-1}}.$$

Remember that we made the choice $X = \frac{1}{2} N^{\frac{1}{\delta} - \epsilon}$. Hence in order to finish the proof of the theorem, it suffices to show that

$$\frac{2m}{\delta m - 1} - \frac{m(1 + \frac{1}{m})}{\delta m - 1} \geq \frac{1}{\delta} - \epsilon.$$

Then multiplying by $\frac{\delta m-1}{\delta m}$ we get :

$$\frac{2}{\delta} - \frac{1}{\delta}(1 + \frac{1}{m}) \geq \frac{1}{\delta} - \epsilon.$$

This simplifies to

$$\frac{-1}{\delta} \frac{1}{m} \geq -\epsilon.$$

and finally it gives us the condition

$$m \geq \frac{1}{\delta \epsilon},$$

which holds because we made the choice $m = \lceil 1/(\delta\epsilon) \rceil$. $\square$

Algorithm 3: Coppersmith method.

```
Input: Polynomial $f(x)$ of degree $\delta$, modulus $N$ of unknown
       factorization and $0 < e < \frac{1}{7}$
Output: Set $R$, where $x_0 \in R$ whenever $f(x_0) = 0 \mod N$ for an
        $|x_0| \leq X$
$m = \lceil 1/(\delta\epsilon) \rceil$
$X = \frac{1}{2} N^{\frac{1}{\delta} - \epsilon}$
for $i \leftarrow 1$ to $m$ do
    for $j \leftarrow 0$ to $\delta - 1$ do
        $g_{i,j}(x) = x^j N^i f^{m-i}(x)$
Construct the lattice basis $B$, where the basis vectors of $B$ are the
coefficient vectors of $g_{i,j}(xX)$.
$\mathbf{v}$ = LLL($B$ , $c = 2$).get_column(0)
Construct $g(x)$ from $\mathbf{v}$.
Find the set $R$ of all roots of $g(x)$ over the integers using standart
methods. For every root $x_0 \in R$ check wether $\gcd(N, f(x_0)) \geq N$. If
it is not the case then remove $x_0$ from $R$.
```

3.3 Application

We can use the coppersmith method in order to attack RSA under certain conditions. We can use it to solve the Relaxed RSA problem where e is small and we have an approximation $\tilde{M}$ of M such that $M = \tilde{M} + x_0$ for some unknown part $|x_0| \leq N^{\frac{1}{e}}$. To solve this problem using Coppersmith method we define

$$f(x) = (\tilde{M} + x)^e - M^e \mod N.$$

And we can recover x_0 applying the coppersmith method to $f(x)$ as long as $x_0 \leq N^{\frac{1}{e}}$.

4 Implementation and examples

To implement and test the algorithms (LLL & Coppersmith), I used Sage. Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface. Its goal is to create a viable free open source alternative to Magma, Maple, Mathematica and Matlab. Website : `http://www.sagemath.org/`

4.1 LLL

As python is an interpreted language (it is not compiled), we don't expect to have good performance with our implementation of LLL. When a lot of computation are required, we prefer the native and optimized sage implementation that is much faster.

Code 1: Computes the Gram-Schmidt orthogonalization and then test if the matrix B is c-reduced or not.

```
def is_LLL_reduced(B, c = 2):
   n = B.nrows()
   m = B.ncols()
   U = matrix(RR, m, m)
   O = matrix(RR, n, m)
   for i in range(0, m) :
      U[i,i] = 1
      O.set_column(i, B.column(i))
      for j in range(0, i) :
         U[j,i] = (B.column(i)*O.column(j))/ \
                  (O.column(j)*O.column(j))
         O.set_column(i, O.column(i) - U[j,i]*O.column(j))

   for i in range(0, m-1) :
      if O.column(i)*O.column(i) > \
         c*O.column(i+1)*O.column(i+1) :
         return False
   return True
```

Code 2: LLL.

```
def reduce(i, B, U):
   j = i-1
   while j >= 0 :
      B.set_column(i, B.column(i) - \
         round(U[j,i])*B.column(j))
      U.set_column(i, U.column(i) - \
         round(U[j,i])*U.column(j))
      j = j - 1

def LLL(B, c = 2):
   n = B.nrows()
   m = B.ncols()
   U = matrix(RR, m, m)
   O = matrix(RR, n, m)
   for i in range(0, m) :
      U[i,i] = 1
      O.set_column(i, B.column(i))
      for j in range(0, i) :
         U[j,i] = (B.column(i)*O.column(j))/ \
                  (O.column(j)*O.column(j))
         O.set_column(i, O.column(i) - U[j,i]*O.column(j))
         reduce(i, B, U)

   i = 0
   while i < m-1 :
   if O.column(i)*O.column(i) <= \
      c*O.column(i+1)*O.column(i+1) :
      i = i + 1
   else :
      O.set_column(i+1, O.column(i+1) + \
         U[i,i+1]*O.column(i))
      U[i,i] = (B.column(i)*O.column(i+1))/ \
               (O.column(i+1)*O.column(i+1))
      U[i+1,i] = 1
      U[i, i+1] = 1
      U[i+1,i+1] = 0
      O.set_column(i, O.column(i)-U[i,i]*O.column(i+1))
      U.swap_columns(i,i+1)
      O.swap_columns(i,i+1)
      B.swap_columns(i,i+1)
      for k in range(i+2, m) :
         U[i,k] = (B.column(k)*O.column(i))/ \
                  (O.column(i)*O.column(i))
         U[i+1, k] = (B.column(k)*O.column(i+1))/ \
                     (O.column(i+1)*O.column(i+1))
      if abs(U[i,i+1]) > 0.5 : reduce(i+1, B, U)
         i = max(i-1,0)
   return B
```

In order to test if the implementation of LLL works properly, and to have an idea of the running time, we run it several times on random matrices of different sizes with the following code :

Code 3: LLL running time test

```
runtimes = []
for i in range(0, 39) :
   runtimes.append(0.0)
   for k in range(0, 10) :
      r = 0
      while r != i+2 :
         A = random_matrix(ZZ, i+2)
         r = A.rank()
      t = cputime()
      res = LLL(A)
      runtimes[i] = runtimes[i] + cputime(t)*0.1
      if is_LLL_reduced(res) == False :
         print("LLL FAILURE")
print(runtimes)
```

With the runtimes list we can plot the running time. Here is the result I got on my computer :

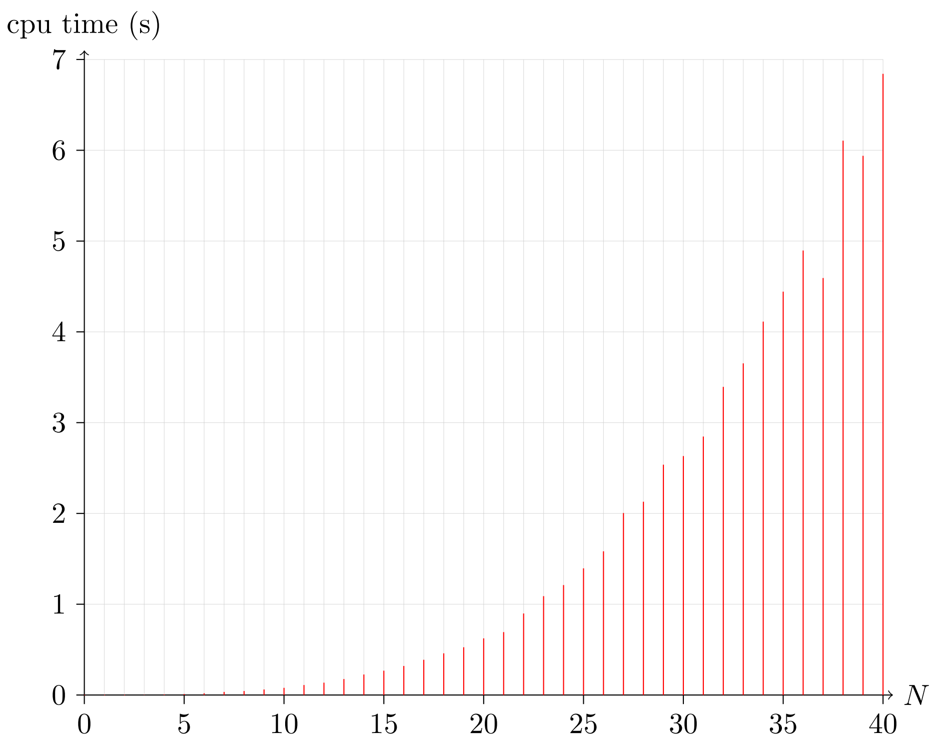

Figure 9: LLL execution time with random matrices of size NxN

We can clearly notice that the runtime is polynomial with respect to the matrix size. The runtime depends also on the basis vectors. For example it is possible that we generate a random matrix that is already reduced, then LLL has nothing to do. Here we compute the running time for each size N doing the average on 10 different matrices of size N. Because 10 is small, it explains why we can get that the running time for a 39x39 matrix is shorter than the one for a 38x38 matrix.

4.2 Coppersmith

Code 4: Coppersmith method

```
def coppersmith(f, N, epsilon = 0.1, fastLLL = False \
                                        , debug = False) :
   if epsilon > 1/7.0 or epsilon <= 0 :
      print("invalid epsilon")
      return None
   f.change_ring(Integers(N))
   delta = f.degree()
   m = ceil(1/delta/epsilon)
   R.<x> = ZZ[]
   #construction of the g[i,j](x)
   g = []
    for j in range(0, delta) :
       g.append([])
       for i in range(1, m+1) :
          g[j].append(x^j*N^(i)*f^(m-i))
   X = ceil(0.5*N^(1/delta - epsilon))
   if debug : print("X = " + str(X))
   size = m*delta
   #construct B from g[i,j](X*x)
   B = matrix(ZZ, size, size)
   compteur = 0
   for i in range(-m+1, 1) :
      for j in range(0, delta) :
         polylist = g[j][-i](X*x).list()
         vector = [0]*size
         vector[0:len(polylist)] = polylist
         vector.reverse()
         B.set_column(compteur, vector)
         compteur = compteur + 1
   if debug : show(B)
   if debug : print "LLL starts"
   coeffs = []
   #computes a small combination of g[i,x](X*x) with LLL
   if fastLLL : #use native sage implementation
      coeffs = B.transpose().LLL().transpose().\
                              column(0).list()
   else  :  #use our python implementation
      coeffs = LLL(B).column(0).list()
   coeffs.reverse()
   #construct g(x)
   g = 0*x
   for i in range(0, size) :
      g = g +  Integer(coeffs[i]/X^i) * x^i
   #get the roots of g(x) over the integers
   roots = g.roots(multiplicities=False)
   result = []
   #test if the roots x_i respect f(x_i) = 0 mod N
   for i in range(0, len(roots)) :
      if gcd(N, f(roots[i])) >= N :
         result.append(roots[i])
   return result
```

Code 5: Coppersmith method example

```
R.<x> = ZZ[]
f = (x-1)*(x-2)*(x-3)*(x-4)*(x-40)
print coppersmith(f, 10000)
```

In this example $X = 2$ and the Coppersmith method outputs the list [2,1].

4.3 RSA attack

To test an attack on RSA using the Coppersmith method, we first need to create RSA keys.

Code 6: RSA key creation : Computes d from p, q, and e.

```
def generate_d(p, q, e) :
   if not is_prime(p) :
      print "p is not prime"
      return None
      if not is_prime(q) :
         print "q is not prime"
         return None
   euler = (p-1)*(q-1)
   if gcd(e, euler) != 1 :
      print "e is not coprime with (p-1)(q-1)"
      return None
   return inverse_mod(e, euler)
```

Code 7: Simple example of an attack

```
p = 17
q = 37
n = p*q
e = 7
d = generate_d(p,q,e)
M = 21
C = power_mod(M, e, n)
MA = 20 #Approximation of M
R.<x> = ZZ[]
f = (MA + x)^e - C
print coppersmith(f, p*q)
```

In this example $X = 1$ and the coppersmith method outputs [1]. It follows that we can recover the message M : $M = MA + 1$

Code 8: Example of an attack

```
p = 955769
q = 650413
```

```
n = p*q
e = 5
d = generate_d(p,q, e)
M = 423909
C = power_mod(M, e, n)
MA = 423919 #Approximation of M
R.<x> = ZZ[]
f = (MA + x)^e - C
print coppersmith(f, p*q, 0.1,True)
```

In this example $X = 8$ and the method coppersmith method outputs [-10]. Notice that since $8 < 10$, it could have failed. In this case we use the LLL implementation of sage, because the coefficients of the $g[i, j](X * x)$ vectors seem to be too large for our naive LLL implementation.

5 Acknowledgements

I thank Bertrand Meyer for his good advice and helpful assistance.

6 Bibliography

References

[1] David C. LAY. *Linear Algebra and its applications.* Pearson Education, Inc, publishing as Addison Wesley Higher Education, 2003.

[2] Hendrik W. Lenstra, Jr. *Lattices.* In Algorithmic number theory: lattices, number fields, curves and cryptography Math. Sci. Res. Inst. Publ. 127-181 Cambridge Univ. Press Cambridge, 2008.

[3] Alexander May. *Using LLL-Reduction for Solving RSA and Factorization Problems.* Department of Computer Science TU Darmstadt.

ETH Zuerich, D-Math

Bachelor Thesis

The LLL-Algorithm and some Applications

Author:
Karin Peter

Supervisor:
Paul-Olivier Dehaye

June 29, 2009

Contents

Introduction

In 1982 Arjen Lestra, Hendrik Lenstra Jr. and Laszlo Lovasz published the **LLL-reduction-algorithm**. It was originally meant to find "short" vectors in lattices, i.e. to determine a so called **reduced Basis** for a given lattice. This algorithm also helped finding solutions to two other major problems: the **factorization of polynomials** and the search for **integer relations**.
On the following pages we will first describe the LLL-Algorithm and derive all its steps. We will then determine the relation between lattice reduction and the problem of factoring polynomials, and the relation between lattice reduction and finding integer relations. We will closely follow the layout of the original paper of Lenstra, Lenstra and Lovasz (see [8]). As an application of integer relation, we are going to discuss **BBP-type formulae** (which have actually not been obtained by the LLL-Algorithm but a more efficient Algorithm, the **PSLQ-Algorithm**).

1 The LLL-algorithm

In this first chapter we will define some expressions and recall the Gram-Schmidt orthogonalization process since it is crucial in the algorithm.
We will then present the LLL-Algorithm and derive all the associated steps. Finally we present a theorem that will give us a few properties on reduced basis of lattices that we will need in chapter two and three.

1.1 Lattices, Gram-Schmidt and some properties

Definition 1.1: A subset L of the real vector space $\mathbb{R}^n$ is called a *lattice* if there exist a basis $b_1, b_2, ...b_n$ of $\mathbb{R}^n$ such that

$$L = \Big\{ \sum_{i=1}^{n} r_i b_i \big| r_i \in \mathbb{Z} \text{ for } i \in \{1, ..., n\} \Big\}$$

We call $b_1, ...b_n$ a *basis* for L and n the *rank* of L.
Moreover we define $d(L) := |\det(b_1, b_2, ...b_n)|$ to be the *determinant* of the lattice.

Gram-Schmidt orthogonalization process: Let $b_1, ..., b_n$ be some linear independent vectors in $\mathbb{R}^n$.

We define inductively:

$$b_1^* = b_1$$

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \text{ for } 1 \leq i \leq n \tag{1}$$

$$\mu_{i,j} = \frac{(b_i, b_j^*)}{|b_j^*|^2} \text{ for } 1 \leq j < i \leq n \tag{2}$$

This process produces vectors $b_1^*, b_2^*, ..., b_n^*$ that form an orthogonal basis of $\mathbb{R}^n$

Definition 1.2: We call a basis $b_1, b_2, ...b_n$ of a lattice L *reduced* if

$$|\mu_{i,j}| \leq \frac{1}{2} \text{ for } 1 \leq j < i \leq n \tag{3}$$

and

$$|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 \geq \frac{3}{4} |b_{i-1}^*|^2 \text{ for } 1 < i \leq n \tag{4}$$

The second condition can be rewritten as $|b_i^*| \geq (\frac{3}{4} - \mu_{i,i-1}^2)|b_{i-1}^*|^2$, which is known as the Lovasz's condition.
Note that the constant $\frac{3}{4}$ in the definition is arbitrary chosen. Indeed, we could take any other constant between $\frac{1}{4}$ and 1.

1.2 The algorithm

We are now going to present an algorithm that takes an arbitrary basis of a lattice as an input and returns a reduced basis of the same lattice. As we see by the definition, we will need an orthogonal basis to check the two properties of reduced bases. So we will first apply Gram-Schmidt orthogonalization process to the basis given. The algorithm then modifies the basis elements such that they fulfill the desired properties. These modifications will be described in detail in the next section.

1.2.1 Implementation of the algorithm

```
1. for $i = 1 : n$
2.     $b_i^* := b_i;$
3.     for $j = 1 : i - 1$
```

```
4.        $\mu_{ij} := (b_i, b_j^*)/(b_j^*, b_j^*)$;
5.        $b_i^* := b_i^* - \mu_{ij} b_j^*$;
6.    end
7. end

8. $k := 2$;

9. $l := k - 1$;
10. if $|\mu_{kl}| > \frac{1}{2}$
11.    $r :=$ integer nearest to $\mu_{kl}$;
12.    $b_k := b_k - r b_l$;
13.    for $j = 1 : l - 1$
14.        $\mu_{kj} := \mu_{kj} - r\mu_{lj}$;
15.    end
16.    $\mu_{kl} := \mu_{kl} - r$;
17. end

18. if $|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 \geq \frac{3}{4}|b_{k-1}^*|^2$
19.    go to (10) and perform for $l = k - 2 : 1$;
20.    if $k = n$, terminate
21.    else $k := k + 1$; go to (9);
22. end
23. else
```

```
24.     $\begin{pmatrix} b_{k-1} \\ b_k \end{pmatrix} := \begin{pmatrix} b_k \\ b_{k-1} \end{pmatrix}$;
25.     $B_{k-1} := b^*_{k-1}$;
26.     $B_k := b^*_k$;
27.     $\mu := \mu_{k,k-1}$;
28.     $b^*_{k-1} := B_k + \mu B_{k-1}$;
29.     $\mu_{k,k-1} := \mu \frac{|B_{k-1}|^2}{|b^*_{k-1}|^2}$;
30.     $b^*_k := B_{k-1} - \mu_{k,k-1} b^*_{k-1}$;
31.     for $j = k+1 : n$
32.         $\begin{pmatrix} \mu_{j,k-1} \\ \mu_{jk} \end{pmatrix} := \begin{pmatrix} \mu_{j,k-1}\mu_{k,k-1} + \mu_{jk}\frac{|B_k|^2}{|B_{k-1}|^2} \\ \mu_{i,k-1} - \mu_{ik}\mu \end{pmatrix}$
33.     end
34.     for $j = 1 : k-2$
35.         $\begin{pmatrix} \mu_{k-1,j} \\ \mu_{kj} \end{pmatrix} := \begin{pmatrix} \mu_{kj} \\ \mu_{k-1,j} \end{pmatrix}$
36.     end
37.     if $k > 2$, $k := k-1$;
38.     go to (9)
39. end
```

1.2.2 Derivation of the steps

Let $\{b_1, b_2, ..., b_n\}$ be a basis of the lattice L. The procedure to receive a reduced basis for L out of $\{b_1, b_2, ..., b_n\}$ is as follows:

For the initialization we first need to calculate all the $\{b^*_1, b^*_2, ..., b^*_n\}$ and μ_{ij} for $1 \leq j <$

$i \leq n$ with the Gram-Schmidt process.
This is an iteration process, therefore we assume the basis is already reduced for $b_1, b_2, ..., b_{k-1}$ for some $k < n$, i.e. the conditions (3) and (4) are fulfilled for this set. The aim is now to modify b_k (and if necessary $b_1, ..., b_{k-1}$) in such a way that we can get a new set $\{b'_1, b'_2, ..., b'_k\}$ which fulfil the conditions of a reduced basis. We will also have to adapt the b^*_i's and μ_{ij} , such that (1) and (2) remains valid.
In every loop we will have to check the two conditions (3) and (4) for the current b_k.

1. **Check if** $|\mu_{k,k-1}| \leq \frac{1}{2}$

   - If $|\mu_{k,k-1}| \leq \frac{1}{2}$, continue with point 2.

   - If $|\mu_{k,k-1}| > \frac{1}{2}$
     Define r to be the nearest integer to $\mu_{k,k-1}$.
     Set $b'_k := b_k - rb_{k-1}$
     Obviously we now have to modify all the μ_{kj}, $1 \leq j < k$ as well.
     Note: $\mu_{kj} = \frac{(b_k, b^*_j)}{|b^*_j|^2}$ and therefore for the new μ'_{kj}'s we have:

     $$\mu'_{kj} = \frac{(b_k - rb_{k-1}, b^*_j)}{|b^*_j|^2} = \frac{(b_k, b^*_j)}{|b^*_j|^2} - r\frac{(b_{k-1}, b^*_j)}{|b^*_j|^2}$$

     hence:

     $$\mu'_{kj} := \mu_{kj} - r\mu_{k-1,j}, \text{ for } 1 \leq j < k-1 \text{ and}$$
     $$\mu'_{k,k-1} := \mu_{k,k-1} - r$$

     If we define r this way, we get that $|\mu_{k,k-1}| \leq \frac{1}{2}$.
     We do not have to modify the b^*_i's for $i \in \{1, ..., n\}$. One can see this with the following equation:

     $$b^*_i = b_i - \sum_{j=1}^{i-1} \mu_{ij} b^*_j$$

     Now only b_k has been modified, so we only have to check for b^*_k (and b^*_i for

$i \in [k+1,\dots,n]$):

$$\begin{aligned}
b_k'^* &= b_k' - \sum_{j=1}^{k-1} \mu_{kj}' b_j'^* \\
&= b_k - rb_{k-1} - \sum_{j=1}^{k-1} (\mu_{kj} - r\mu_{k-1,j}) b_j^* \\
&= b_k - \sum_{j=1}^{k-1} \mu_{kj} b_j^* - r\big(b_{k-1} - \sum_{j=1}^{k-1} \mu_{k-1,j} b_j^*\big) \\
&= b_k^* - r(b_{k-1}^* - \mu_{k-1,k-1} b_{k-1}^*) \\
&= b_k^*
\end{aligned}$$

Since the b_i^*'s are not changed, also the μ_{jk} for $j > k$ are not changed. We see this with the definition of μ_{jk}'.

$$\mu_{jk}' := \frac{(b_j', b_k'^*)}{|b_k'^*|^2} = \frac{(b_j, b_k^*)}{|b_k^*|^2} = \mu_{jk}$$

Note: It is still not guaranteed that $\mu_{kj} \leq \frac{1}{2}$, for j smaller than $k-1$.

2. **Check if** $|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 \geq \frac{3}{4}|b_{k-1}^*|^2$

- If $|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 \geq \frac{3}{4}|b_{k-1}^*|^2$, focus on the μ_{kj}, $1 \leq j < k-1$
  Let l be the largest index, s.t. $\mu_{kl} > \frac{1}{2}$.
  Define r to be the nearest integer to μ_{kl}.
  Set $b_k := b_k - rb_l$.
  As in point 1 we now have to modify the μ_{kj}, $1 \leq j \leq l$. With the same calculation we get:

$$\begin{aligned}
&\mu_{kj}' := \mu_{kj} - r\mu_{l,j}, \text{ for } 1 \leq j < l-1 \text{ and} \\
&\mu_{k,l}' := \mu_{k,l} - r
\end{aligned}$$

Note: We don't have to modify μ_{kj}, $l < j < k$ because:

$$\begin{aligned}
\mu'_{kj} &= \frac{(b_k - rb_l, b_j^{'*})}{|b_j^{'*}|^2} \\
&= \frac{(b_k, b_j^*)}{|b_j^*|^2} - r\frac{(b_l, b_j^*)}{|b_j^*|^2} \\
&= \mu_{kj} - r\frac{(b_l^* - \sum_{i=1}^{l-1} \mu_{li} b_i^*, b_j^*)}{|b_j^*|^2} \\
&= \mu_{kj} - r\Big(\frac{(b_l^*, b_j^*)}{|b_j^*|^2} + \sum_{i=1}^{l-1} \mu_{li}\frac{(b_i^*, b_j^*)}{|b_j^*|^2}\Big) \\
&= \mu_{kj}.
\end{aligned}$$

because of the orthogonality of the b_i^*'s, and $j > l$.

Since this is not going to change the b_i^*'s, (4) remain valid and we can repeat this procedure until $\mu_{kj} \leq \frac{1}{2}$ for all $1 \leq j < k-1$. Now we can replace k by $k+1$ and continue with the first step.

- If $|b_k^* + \mu_{k,k-1} b_{k.1}^*|^2 < \frac{3}{4}|b_{k-1}^*|^2$, we need to modify the b_i's.
  Interchange b_k and b_{k-1}, i.e. set $b'_k := b_{k-1}$ and $b'_{k-1} := b_k$.
  We now have to modify b_k^* and b_{k-1}^* (the b_i^* for $i \neq k, k-1$ do not change):

$$\begin{aligned}
b_{k-1}^{'*} &= b'_{k-1} - \sum_{j=1}^{k-2} \mu'_{k-1,j} b_j^{'*} \\
&= b_k - \sum_{j=1}^{k-2} \mu_{k,j} b_j^* \\
&= b_k - \sum_{j=1}^{k-1} \mu_{kj} b_j^* + \mu_{k,k-1} b_{k-1}^* \\
&= b_k^* + \mu_{k,k-1} b_{k-1}^*
\end{aligned} \tag{5}$$

$$\begin{aligned}
b_k^{'*} &= b'_k - \sum_{j=i}^{k-1} \mu'_{kj} b_j^{'*} \\
&= b_{k-1} - \sum_{j=1}^{k-2} \mu_{k-1,j} b_j^* - \mu'_{k,k-1} b_{k-1}^{'*} \\
&= b_{k-1}^* - \mu'_{k,k-1} b_{k-1}^{'*} \\
&= b_{k-1}^* - \frac{(b'_k, b_{k-1}^{'*})}{|b_{k-1}^{'*}|^2} b_{k-1}^{'*} \\
&= b_{k-1}^* - \frac{(b_{k-1}, b_k^* + \mu_{k,k-1} b_{k-1}^*)}{|b_{k-1}^{'*}|^2} b_{k-1}^{'*} \\
&= b_{k-1}^* - \frac{(b_{k-1}^* + \sum_{i=1}^{k-2} \mu_{k-1,i} b_i^*, b_k^* + \mu_{k,k-1} b_{k-1}^*)}{|b_{k-1}^{'*}|^2} b_{k-1}^{'*} \\
&= b_{k-1}^* - \frac{(b_{k-1}^*, \mu_{k,k-1} b_{k-1}^*)}{|b_{k-1}^{'*}|^2} \\
&= b_{k-1}^* - \mu_{k,k-1} \frac{|b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2} b_{k-1}^{'*} \\
&= b_{k-1}^* - \mu'_{k,k-1} b_{k-1}^{'*} \qquad (6)
\end{aligned}$$

Where in the seventh line of the calculation of $b_k^{'*}$ we used the orthogonality of the b_i^*'s.

Now we need to modify $\mu_{i,k-1}$ and $\mu_{i,k}$ for $i > k$, $\mu_{k-1,i}$ and μ_{ki} for $1 \leq i < k-1$.
First note that $b_i = b'_i$ for $i > k$, hence with (1) we have:

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* = b_i^{'*} + \sum_{j=1}^{i-1} \mu'_{ij} b_j^{'*} = b'_i$$

Since $b_j^{'*} = b_j^*$ for $j \neq k, k-1$ we can subtract the common terms on both sides of the equation to get:

$$\mu_{i,k-1} b_{k-1}^* + \mu_{ik} b_k^* = \mu'_{i,k-1} b_{k-1}^{'*} + \mu'_{ik} b_k^{'*} \qquad (7)$$

From (6) we have:

$$b_{k-1}^* = b_k^{'*} + \mu'_{k,k-1} b_{k-1}^{'*}$$

and from (5),

$$\begin{aligned}
b_k^* &= b_{k-1}^{'*} - \mu_{k-1} b_{k-1}^* \\
&= b_{k-1}^{'*} - \mu_{k,k-1}(b_k^{'*} + \mu'_{k,k-1} b_{k-1}^{'*}) \\
&= (1 - \mu_{k,k-1}\mu'_{k,k.1}) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \mu_{k,k-1}^2 \frac{|b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \frac{(b_k, b_{k-1}^*)^2}{|b_{k-1}^*|^4} \frac{|b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \frac{(b_k, b_{k-1}^*)^2}{|b_{k-1}^*|^2 |b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \frac{(b_k^* + \sum_{i=1}^{k-1} \mu_{ki} b_i^*, b_{k-1}^*)^2}{|b_{k-1}^*|^2 |b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \frac{(\mu_{k,k-1} b_{k-1}^*, b_{k-1}^*)^2}{|b_{k-1}^*|^2 |b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(1 - \frac{\mu_{k,k-1}^2 |b_{k-1}^*|^4}{|b_{k-1}^*|^2 |b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(\frac{|b_{k-1}^{'*}|^2 - \mu_{k,k-1}^2 |b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&\overset{(5)}{=} \Big(\frac{|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 - \mu_{k,k-1} |b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \Big(\frac{|b_k^*|^2 + \mu_{k,k-1}^2 |b_{k-1}^*|^2 - \mu_{k,k-1}^2 |b_{k-1}^*|^2}{|b_{k-1}^{'*}|^2}\Big) b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*} \\
&= \frac{|b_k^*|^2}{|b_{k-1}^{'*}|^2} b_{k-1}^{'*} - \mu_{k,k-1} b_k^{'*}
\end{aligned}$$

So with (7) we have :

$$\mu_{i,k-1}(b_k^{'*} + \mu'_{k,k-1} b_{k-1}^{'*}) + \mu_{ik}\Big(\frac{|b_k^*|^2}{|b_{k-1}^{'*}|^2} - \mu_{k,k-1} b_k^{'*}\Big) = \mu'_{i,k-1} b_{k-1}^{'*} + \mu'_{ik} b_k^{'*}$$

$$\iff (\mu_{i,k-1} - \mu_{ik}\mu_{k,k-1}) b_k^{'*} + (\mu_{i,k-1}\mu'_{k,k-1} + \mu_{ik}\frac{|b_k^*|^2}{|b_{k-1}^*|^2}) b_{k-1}^{'*} = \mu'_{i,k-1} b_{k-1}^{'*} + \mu'_{ik} b_k^{'*}$$

$$\Longrightarrow \begin{cases} \mu'_{i,k-1} = \mu_{i,k-1}\mu'_{k,k-1} + \mu_{ik}\frac{|b_k^*|^2}{|b_{k-1}^*|^2} \\ \mu'_{ik} = \mu_{i,k-1} - \mu_{ik}\mu_{k,k-1} \end{cases}$$

For $\mu'_{k-1,j}$ and μ'_{kj}, $1 \leq j < k-1$ one can easily see that:

$$\mu'_{k-1,j} = \mu_{kj}$$
$$\mu'_{kj} = \mu_{k-1,j}$$

Now we can replace k by $k-1$ and continue with the first step.

1.2.3 Proof

Claim: The algorithm described above terminates and it returns a reduced basis $b'_1, b'_2, ..., b'_n$ of L.

Proof. Obviously the vectors $b'_1, b'_2, ..., b'_k$ are reduced after every loop, so we only have to prove that the algorithm terminates.
We had that every time we interchanged b_k and b_{k-1} we lowered the current index by one. In every other case we increased it by one. So we need to show that there are only finitely many times where we need to interchange b_k and b_{k-1}.
In order to show this, we define

$$d_i := \det(D_i) \text{ with } D_i := (b_j, b_l)_{1 \leq j,l \leq i}$$
$$D := \prod_{i=1}^{n-1} d_i$$

We now need to compute the effect of the changes of the b_i's in the algorithm on the two quantities d_i and D.
We first derive that $d_i = \prod_{j=1}^{i} |b_j^*|^2$, for all $i \in \{1, ..., n\}$.
One can see this by applying the Gaussian elimination procedure to D_i. As we know, the determinant will not change under these operations.
We have:

$$d_i = \det(D_i) = \det \begin{pmatrix} (b_1,b_1) & (b_1,b_2) & \dots & (b_1,b_i) \\ (b_2,b_1) & (b_2,b_2) & \dots & (b_2,b_i) \\ \vdots & & & \vdots \\ (b_i,b_1) & & \dots & (b_i,b_i) \end{pmatrix} \tag{8}$$

The first step is to subtract $\frac{(b_j,b_1)}{(b_1,b_1)}$ times the first row from the j-th row, for all $j \in \{2, ..., i\}$. As a property of the Gaussian elimination we get zeros on each entry of the first column except in the first row. The first row remains unchanged and for the k-th entry ($k \in \{2, ...i\}$) of the j-th row we then get:

$$(b_j, b_k) - \frac{(b_j, b_1)}{(b_1, b_1)}(b_1, b_k) = (b_j - \frac{(b_j, b_1^*)}{|b_1^*|^2} b_1^*, b_k) = (b_j - \mu_{j1} b_1^*, b_k)$$

Since we have $b_2 - \mu_{21}b_1^* = b_2^*$ and $(b_j^*, b_j) = (b_j^*, b_j^* + \sum_{i=1}^{j-1} \mu_{ji}b_i^*) = |b_j^*|^2$ we get:

$$d_i = \det \begin{pmatrix} |b_1^*|^2 & (b_1^*, b_2) & (b_1^*, b_3) & \dots & (b_1^*, b_i) \\ 0 & |b_2^*|^2 & (b_2^*, b_3) & \dots & (b_2^*, b_i) \\ 0 & (b_3 - \mu_{31}b_1^*, b_2) & (b_3 - \mu_{31}b_1^*, b_3) & \dots & (b_3 - \mu_{31}b_1^*, b_i) \\ \vdots & & & & \vdots \\ 0 & (b_i - \mu_{i1}b_1^*, b_2) & & \dots & (b_i - \mu_{i1}b_1^*, b_i) \end{pmatrix}$$

The second step is then to subtract $\frac{(b_j - \mu_{j1}b_1^*, b_2)}{|b_2^*|^2}$ times the second row from the j-th row ($j \in \{3, ..., i\}$).
For the k-th entry ($k \in \{3, ..., i\}$) of the j-th row we then get:

$$\begin{aligned} &(b_j - \mu_{j1}b_1^*, b_k) - \frac{(b_j - \mu_{j1}b_1^*, b_2)}{|b_2^*|^2}(b_2^*, b_k) \\ &= (b_j - \mu_{j1}b_1^* - \frac{(b_j - \mu_{j1}b_1^*, b_2)}{|b_2^*|^2}b_2^*, b_k) \\ &= (b_j - \mu_{j1}b_1^* - \frac{(b_j - \mu_{j1}b_1^*, b_2^* + \mu_{21}b_1^*)}{|b_2^*|^2}b_2^*, b_k) \\ &= b_j - \mu_{j1}b_1^* - \left(\frac{(b_j, b_2^*)}{|b_2^*|^2} - \frac{(\mu_{j1}b_1^*, b_2^*)}{|b_2^*|^2} + \frac{(b_j, \mu_{21}b_1^*)}{|b_2^*|^2} - \frac{(\mu_{j1}b_1^*, \mu_{21}b_1^*)}{|b_2^*|^2}\right) b_2^*, b_k) \\ &= b_j - \mu_{j1}b_1^* - \left(\mu_{j2} + \frac{(b_j, \mu_{21}b_1^*)}{|b_2^*|^2} - \frac{(\frac{(b_j, b_1^*)}{|b_1^*|^2}b_1^*, \mu_{21}b_1^*)}{|b_2^*|^2}\right) b_2^*, b_k) \\ &= b_j - \mu_{j1}b_1^* - \left(\mu_{j2} + \frac{(b_j, \mu_{21}b_1^*)}{|b_2^*|^2} - \frac{\frac{(b_j, b_1^*)}{|b_1^*|^2}\mu_{21}(b_1^*, b_1^*)}{|b_2^*|^2}\right) b_2^*, b_k) \\ &= b_j - \mu_{j1}b_1^* - \left(\mu_{j2} + \frac{(b_j, \mu_{21}b_1^*)}{|b_2^*|^2} - \frac{(b_j, b_1^*)\mu_{21}}{|b_2^*|^2}\right) b_2^*, b_k) \\ &= (b_j - \mu_{j1}b_1^* - \mu_{j2}b_2^*, b_k) \end{aligned}$$

Hence,

$$d_i = \det \begin{pmatrix} |b_1^*|^2 & (b_1^*, b_2) & (b_1^*, b_3) & \dots & (b_1^*, b_i) \\ 0 & |b_2^*|^2 & (b_2^*, b_3) & \dots & (b_2^*, b_i) \\ 0 & 0 & |b_3^*|^2 & \dots & (b_3^*, b_i) \\ 0 & 0 & (b_4 - \mu_{41}b_1^* - \mu_{42}b_2^*, b_3) & \dots & (b_4 - \mu_{41}b_1^* - \mu_{42}b_2^*, b_i) \\ \vdots & & & & \vdots \\ 0 & 0 & (b_i - \mu_{i1}b_1^* - \mu_{i2}b_2^*, b_3) & \dots & (b_i - \mu_{i1}b_1^* - \mu_{i2}b_2^*, b_i) \end{pmatrix}$$

Apparently by proceeding with the Gaussian elimination, we get

$$d_i = \det \begin{pmatrix} |b_1^*|^2 & (b_1^*, b_2) & (b_1^*, b_3) & \dots & (b_1^*, b_i) \\ 0 & |b_2^*|^2 & (b_2^*, b_3) & \dots & (b_2^*, b_i) \\ 0 & 0 & |b_3^*|^2 & \dots & (b_3^*, b_i) \\ \vdots & & & & \vdots \\ 0 & \dots & & & |b_i^*|^2 \end{pmatrix} = \prod_{j=1}^{i} |b_j^*|^2.$$

Now, only in the case when $|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 < |b_{k-1}^*|^2$ we had to modify some of the b_i^*'s (namely b_{k-1}^* and b_k^*). We had $|b_{k-1}^{\prime *}|^2 = |b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 < \frac{3}{4}|b_{k-1}^*|^2$, hence d_{k-1} decrease by a factor $< \frac{3}{4}$ with every such change being made. All the other d_j, $j \in \{1, ..., i\} \setminus \{k-1\}$ remain unchanged as we see with (8).
Since we lower the current index of the algorithm if and only if such a change is made, there is a one-to-one correspondence between the lowering of the current index and the quantity D. That is, if the determinant defined above after one modification of the b_i^*'s is denoted by D', then we have $D' < \frac{3}{4}D$.

Claim: There is a positive lower bound for d_i depending only on L.

Proof. We define $m(L) := \min\{|x|^2; x \in L, x \neq 0\}$, which is a positive, real number. Note that it is not claimed that the LLL-algorithm does find such a shortest vector. For the determinant of the lattice we have

$$d(L) = |\det(b_1, b_2, ...b_n)| = |\det(b_1^*, b_2^*, ...b_n^*)| = \prod_{i=1}^{n} |b_i^*|$$

since the b_i^* are pairwise orthogonal, so $d_n = \det(L)^2$ and d_i for $1 \leq i < n$ is equal to de determinant of the lattice spanned by $b_1, \dots, b_i$. As a property of lattices there exists a vector $x \neq 0$ such that $|x| \leq \frac{4}{3}^{(i-1)/2} d_i^{1/i}$ (see [5, lemma 4, pp. 21 and theorem 1, pp. 31]) and therefore

$$d_i \geq \frac{3}{4}^{(i-1)i/2} |x|^{2i} \geq \frac{3}{4}^{(i-1)i/2} m(L)^i,$$

which is what we wanted. □

So since we lower D by $\frac{3}{4}$ every time we interchange b_k and b_{k-1} and there is a lower bound for D, we conclude that there can only be finitely many interchanges during the algorithm and therefore the algorithm terminates. □

1.3 Properies of reduced bases

Theorem 1.1: If $b_1, b_2, ..., b_n$ is some reduced basis for a lattice L in $\mathbb{R}^n$, then

1. $|b_j|^2 \leq 2^{i-1}|b_i^*|$ for $1 \leq j \leq i \leq n$
2. $d(L) \leq \prod_{i=1}^n |b_i| \leq 2^{n(n-1)/4} d(L)$
3. $|b_1| \leq 2^{(n-1)/4} d(L)^{1/n}$
4. For any linearly independent set of vectors $x_1, x_2, ..., x_t \in L$ we have $|b_j| \leq 2^{(n-1)/2} \max(|x_1|, |x_2|, ..., |x_t|)$ for $1 \leq j \leq t$.

Proof. 1. First note:

$$|b_i|^2 = \left|b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*\right|^2 = |b_i^*|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 |b_j^*|^2 \tag{9}$$

from (1) and since the b_i^*'s are orthogonal for $1 \leq i \leq n$.
So we have the following inequality:

$$\begin{aligned} |b_i|^2 &= |b_i^*|^2 + \mu_{i,i-1}^2 |b_{i-1}^*|^2 + ... + \mu_{i,1}^2 |b_1^*|^2 \\ &\leq |b_i^*|^2 + \frac{1}{4}|b_{i-1}^*| + ... + \frac{1}{4}|b_1^*|^2 \end{aligned}$$

Moreover we have $|b_j^*|^2 \geq \frac{1}{2}|b_{j-1}^*|^2$ by Lovasz's condition, hence $|b_j^*|^2 \leq 2^{i-j}|b_i^*|^2$ for all $j \leq i$. Hence for all i:

$$\begin{aligned} |b_i|^2 &\leq \big(1 + \frac{1}{4}(2 + 2^2 + ... + 2^{i-1})\big)|b_i^*|^2 \\ &= \big(1 + \frac{1}{4}\sum_{j=i}^{i-1} 2^i\big)|b_i^*|^2 \\ &= \big(1 + \frac{1}{4}(2 \cdot 2^{i-1} - 2)\big)|b_i^*|^2 \\ &= \frac{2^{i-1}+1}{2}|b_i^*|^2 \\ &\leq 2^{i-1}|b_i^*|^2 \end{aligned}$$

Therefore:

$$|b_j|^2 \leq 2^{j-1}|b_j^*|^2 \leq 2^{i-j}2^{j-1}|b_i^*|^2 = 2^{i-1}|b_i^*|^2 \tag{10}$$

For $1 \leq j < i \leq n$. This proves part 1.

2. For the second part we need that $d(L) = \prod_{i=1}^n |b_i^*|$ as shown above. From (9) we have that $|b_i^*| \leq |b_i|$ and hence

$$d(L) = \prod_{i=1}^n |b_i^*| \leq \prod_{i=1}^n |b_i| \leq \prod_{i=1}^n 2^{(i-1)/2}|b_i^*| = 2^{n(n-1)/4}\prod_{i=1}^n |b_i^*| = 2^{n(n-1)/4}d(L),$$

where we used the calculations from the proof of the first point of the theorem and that $\prod_{i=1}^n 2^{(i-1)/2} = 2^{\sum_{i=1}^n (i-1)/2} = 2^{n(n-1)/4}$.
This proves part 2.

3. For part 3, we use the equation $|b_j^*|^2 \leq 2^{i-j}|b_i^*|^2$ with $j = 1$ (i.e $|b_1^*|^2 = |b_1|^2 \leq 2^{i-1}|b_i^*|^2$) to get:

$$|b_1|^n \leq \prod_{i=1}^n 2^{(i-1)/2}|b_i^*| = 2^{n(n-1)/4}d(L)$$

Hence $|b_1| \leq 2^{(n-1)/4}d(L)^{1/n}$.

4. Note that we can write $x_j = \sum_{i=1}^n r_{ij}b_i = \sum_{i=1}^n r'_{ij}b_i^*$ with $r_{ij} \in \mathbb{Z}$ since $x_j \in L$ and the b_i's are a lattice basis of L. Let $1 \leq j \leq n$ be fixed and let k be the largest index such that $r_{kj} \neq 0$.
Then we have that $r_{kj} = r'_{kj}$. To see this consider the following calculation:

$$x_j = \sum_{i=1}^k r_{ij}b_i = \sum_{i=1}^k r'_{ij}(b_{ij} - \sum_{l=1}^{i-1} \mu_{il}b_l^*)$$

Where in the second term of the RHS b_k does not appear in the summation and since the b_i's are linearly independent we need r'_{kj} to be r_{kj}.
By the orthogonality of the b_i^*'s we have that $|x_j| = \sum_{i=1}^k |r'_{ij}||b_i^*| \geq |r'_{lj}||b_l^*|$ for all $l \in \{1, ..., k\}$, hence

$$|x_j|^2 \geq |r'_{kj}|^2|b_k^*|^2 \geq |b_k^*|^2.$$

since $r'_{kl} = r_{kl}$ and r_{kl} is an integer. So with (10) we have

$$|b_i|^2 \leq 2^{k-1}|b_k^*|^2 \leq 2^{k-1}|x_j|^2 \leq 2^{n-1}\max(|x_1|^2, |x_2|^2, ..., |x_t|^2) \tag{11}$$

for $i \in \{1, ..., k\}$ since $k \leq n$. Moreover since $t \leq k$ this inequality holds for all $i \in \{1, ..., t\}$, which is what we wanted.

□

2 Factorization of polynomials

By the use of the LLL-Lattice reduction Algorithm we can now create an algorithm to factor arbitrary polynomials $f \in \mathbb{Z}[X]$.

A first observation will be that the algorithm only works for polynomials with no multiple roots. So in the final algorithm we will first have to check that property and modify the polynomial such that it has no multiple roots. This will be done by the use of **resultants**. As a second step we need to know all the irreducible factors of $f \mod p$ where p is some prime. These factors can be determined with the use of **Berlekamp's Algorithm**.
We are going to show some properties of factors of f and $f \mod p$ that will allow us to determine one irreducible factor h_0 by knowing one irreducible factor $h \mod p$ of $f \mod p$.

2.1 The setting

In this section we will have a fixed polynomial $f \in \mathbb{Z}[X]$ of degree n, which will be the polynomial we want to factor out. Moreover we have a polynomial $h \in \mathbb{Z}[X]$ which is an irreducible divisor of f if taken modulo some integer (for the exact definition of h see below). Out of h we are going to find an irreducible factor h_0 of f by use of the LLL-algorithm. So we are also going to need a lattice where every polynomial can be represented by an element of that lattice.

We define h the following way: $\deg(h) = l$ where $l < n$ and h fulfills:

$$h \text{ is monic (i.e has leading coefficient one)} \tag{12}$$
$$(h \mod p^k) \text{ divides } (f \mod p^k) \text{ in } (\mathbb{Z}/p^k\mathbb{Z})[X] \tag{13}$$
$$(h \mod p) \text{ is irreducible in } (\mathbb{Z}/p\mathbb{Z})[X] \tag{14}$$
$$(h \mod p)^2 \text{ does not divide } (f \mod p) \text{ in } (\mathbb{Z}/p\mathbb{Z})[X] \tag{15}$$

Note: From (13) we get that $(h \mod p)$ divides $(f \mod p)$. We see this by considering that if x divides $y \Rightarrow (x \mod p)$ divides $(y \mod p)$ and $((h \mod p^k) \mod p) = (h \mod p)$.

Proposition 2.1: Let f and h be as above. Then there is a polynomial $h_0 \in \mathbb{Z}[X]$ such that h_0 is an irreducible factor of f, $(h \mod p)$ divides $(h_0 \mod p)$ and h_0 is unique up to sign.
Moreover if g divides f in $\mathbb{Z}[X]$, then the following are equivalent:

1. $(h \mod p)$ divides $(g \mod p)$ in $(\mathbb{Z}/p\mathbb{Z})[X]$
2. $(h \mod p^k)$ divides $(g \mod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$
3. h_0 divides g in $\mathbb{Z}[X]$

Proof. The existance of such an h_0 follows from (14) and the note above. If h itself is a divisor of f, then $h_0 = h$, and irreducibility follows form (14). If h does not divide f in $\mathbb{Z}[X]$, then there is a irreducible factor h_0 such that ($h_0 \mod p$) factors into ($h \mod p$) and ($\tilde{h} \mod p$) in $\mathbb{Z}/p\mathbb{Z}$. From (15) we get uniqueness.

2 $\Rightarrow$ 1: Obvious for the same reason as in the note above.

3 $\Rightarrow$ 1: h_0 divides $g \Rightarrow (h_0 \mod p)$ divides $(g \mod p) \Rightarrow (h \mod p)$ divides $(g \mod p)$

1 $\Rightarrow$ 3: Since ($h \mod p$) divides ($g \mod p$) and ($h \mod p$) is no multiple divisor of ($f \mod p$) we have that ($h \mod p$) does not divide ($f/g \mod p$) in $\mathbb{Z}/p\mathbb{Z}[X]$. We have that ($h \mod p$) is a factor of ($h_0 \mod p$), therefore also ($h_0 \mod p$) does not divide ($f/g \mod p$) and h_0 does not divide f/g. Hence h_0 has to be a divisor of g.

3 $\Rightarrow$ 2: From the fact that h_0 divides g we have that ($h_0 \mod p$) divides ($g \mod p$) and hence ($h \mod p$) divides ($g \mod p$). From (14) it follows that ($h \mod p$) and ($f/g \mod p$) have no common divisor in $\mathbb{Z}/p\mathbb{Z}$.
Recall that for two numbers a and b with $\gcd(a, b) = 1$ we have that there exists integers λ and μ such that $a\lambda + b\mu = 1$. We can apply this to ($h \mod p$) and ($f/g \mod p$), i.e. there exists λ and $\mu \in \mathbb{Z}[X]$ such that:

$$(\lambda \mod p)(h \mod p) + (\mu \mod p)(f/g \mod p) = 1$$

$$\Longrightarrow \lambda h + \mu f/g = 1 + \nu p$$

with $\nu \in \mathbb{Z}[X]$.
By multiplying boths sides with g and $v(\nu) = 1 + p\nu + p^2\nu^2 + \ldots + p^{k-1}\nu^{k-1}$ we get:

$$\lambda g v(\nu) h + \mu v(\nu) f = (1 - p\nu) v(\nu) g = (1 - p^k \nu^k) g$$

$$\left(\tilde{\lambda} h + \tilde{\mu} f\right) \mod p^k = g \mod p^k$$

Now since we know that ($f \mod p^k$) is divisible by ($h \mod p^k$), the left hand side is divisible by ($h \mod p^k$) then so is the right hand side, i.e. ($g \mod p^k$) is divisible by ($h \mod p^k$) which is what we wanted. □

Note that if we choose g to equal h_0, the third statement is true and by the equivalence of the three stratements we get that ($h \mod p^k$) divides ($h_0 \mod p^k$).

With all the quantities defined as above, our goal is now to find a way to calculate h_0. This is where the lattice reduction algorithm comes into play.

Defining the lattice: In order to apply the results from the last preceding chapter we need to introduce a lattice L representing all possible polynomials for h_0. Let m be the

dimension of the lattice. Clearly $m \geq l$ because the degree of h_0 is greater or equal to the degree of h. An upper bound for m would be $n-1$ since a (not necessary irreducible) factor of a polynomial has at most degree one less that the polynomial itself. We are going to set the actual value of m later, but anyway we consider it fixed.
We set L to be the set of all polynomials in $\mathbb{Z}[X]$ with the property that if taken modulo p then they are divisible by $(h \mod p)$ in $\mathbb{Z}/p\mathbb{Z}$. Recall that we can identify a polynomial of degree m with a vector in $\mathbb{R}^{m+1}$ ($p(x) = p_m x^m + p_{m-1}x^{m-1} + \ldots + p_0 \longleftrightarrow (p_0, p_1, \ldots, p_m)^T$).
A basis of L is given by: $\{p^k X^i; 0 \leq i < l\} \cup \{hX^j; 0 \leq j \leq m-l\}$.
One can see this by considering that $(h \mod p)$ has to divide each of these polynomials modulo p. Obviously h divides hX^j, hence $(h \mod p)$ divides $(hX^j \mod p)$ as well. The polynomial in the first set are zero when taken modulo p and zero can be divided by everything. Certainly also linear combinations of these basis elements then fulfil the desired property. To see that these two sets indeed cover all the polynomials note that there are $l + (m-l+1) = m+1$ elements in that basis of L and that they are linearly independent.
Furthermore we can calculate the determinant of the lattice L:

$$d(L) = \det \begin{pmatrix} p^k & 0 & \ldots & 0 & h_0 & 0 & \ldots & 0 \\ 0 & p^k & \ldots & 0 & h_1 & h_0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \\ 0 & 0 & \ldots & p^k & h_{l-1} & h_{l-2} & \ldots & \\ 0 & 0 & \ldots & 0 & 1 & h_{l-1} & \ldots & \\ \vdots & \vdots & & 0 & 0 & 1 & \ldots & \\ & & & \vdots & \vdots & & \ddots & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & & 1 \end{pmatrix} = p^{kl}$$

We now going to show that h_0 can be calculated as the greatest common divisor of some basis elements of a reduced basis of L. For this we first need three propositions.

Proposition 2.2: Let $b \in L$ such that $p^{kl} > |f|^m |b|^n$.
Then we have that h_0 divides b in $\mathbb{Z}[X]$ and therefore $\gcd(f, b) \neq 1$.

Proof. Define $g := \gcd(f, b)$. We claim that $(h \mod p)$ divides $(g \mod p)$, hence with proposition 1.2 it follows that h_0 divides g and hence h_0 divides b.
To prove the claim, assume in contrary that $(h \mod p)$ does not divide $(g \mod p)$. Since $(h \mod p)$ is irreducible in $\mathbb{Z}/p\mathbb{Z}$ we have that $(h \mod p)$ and $(g \mod p)$ are relatively prime in $\mathbb{Z}/p\mathbb{Z}$ and therefore

$$\lambda h + \mu g = 1 + \nu p \tag{16}$$

for some $\lambda, \mu, \nu \in \mathbb{Z}[X]$.
Let now $\deg(g) = e$ and $\deg(b) = e'$ and define M to be the set:

$$M = \{\lambda f + \mu b; \lambda, \mu \in \mathbb{Z}[X], \deg(\lambda) < e' - e, \deg(\mu) < n - e\}$$

Obviously M is a subset of the set of all polynomial with integer coefficients and degree lesser or equal to $n + e' - e - 1$, i.e $M \subset (\mathbb{Z} + \mathbb{Z}X + \ldots + \mathbb{Z}X^{n+e'-e-1})$. Furthermore we can see that we can define a basis for M such that it is a lattice of rank $n + e' - 2e$ by a projection as follows:
Define M' to be the projection of M on $(\mathbb{Z}X^e + \mathbb{Z}X^{e+1} + \ldots + \mathbb{Z}X^{n+e'-e-1})$.
Now we can show that the kernel of this projection is trivial and therefore its image has the same rank as M itself. Suppose $(\lambda f + \mu b) \in M$ projects to 0 in M'. Then we have that $\deg(\lambda f + \mu b) < e$. Since g divides f and b, it also divides $(\lambda f + \mu b)$ and we get that $(\lambda f + \mu b) = 0$ and hence $\lambda(f/g) = -\mu(b/g)$. From $g = \gcd(f, b)$ it follows that (f/g) and (b/g) has no common divisor and thus (f/g) has to divide μ. Again with an analysis of the degrees of (f/g) and μ we see that μ needs to be zero, and also λ needs to be zero, which proves that the kernel is trivial.
We then have that the projections of

$$\{X^i f; 0 \leq i < e' - e\} \cup \{X^j b; 0 \leq j < n - e\}$$

on M' are linearly independent and span M'. Furthermore M' is a lattice of dimension $n + e' - 2e$ and from the second point of Theorem 1.1 we get that:

$$d(M') \leq \prod_{i=0}^{e'-e-1} |X^i f| \prod_{j=0}^{n-e-1} |X^j b| = |f|^{e'-e} |b|^{n-e} \leq |f|^m |b|^n < p^{kl} \tag{17}$$

In order to derive a contradiction we are now going to observe that the set $\{\theta \in M; \deg(\theta) < e + l\}$ is a subset of $p^k\mathbb{Z}[X]$.
We choose θ to be an element of this set. By the definition of M, g divides θ. We can multiply the equation (16) by (θ/g) and $v(\nu) = 1 + p\nu + p^2\nu^2 + \ldots + p^{k-1}\nu^{k-1}$ (see also proof of Proposition 2.1) to receive

$$(\tilde{\lambda} h + \tilde{\mu}\theta) \mod p^k = (\theta/g) \mod p^k$$

With $\tilde{\lambda} = \lambda(\theta/g)v(\nu)$ and $\tilde{\mu} = \mu v(\nu)$ and hence both in $\mathbb{Z}[X]$. Since $\theta \in M$, it is the sum of a multiple of f and a multiple of b, and $b \in L$ (in particular $(h \mod p^k)$ divides b) we have that $(h \mod p^k)$ divides θ. So with the equality above we also have that $(h \mod p^k)$ divides $(\theta/g) \mod p^k$. By looking at the degrees, $\deg(h \mod p^k) = l$ and $\deg((\theta/g) \mod p^k) < e + l - e = l$ we see that $((\theta/g) \mod p^k)$ has to be zero, hence also $(\theta \mod p^k)$ has to be zero, which is what we wanted to show.

Now we choose a basis $b_e, b_{e+1}, \ldots, b_{n+e'-e-1}$ of M' such that $\deg(b_i) = i$. Then the matrix representing M' has upper triangular form and we can calculate $d(M')$ very easily by just multiplying the leading coefficients. By the observation above ($\{\theta \in M; \deg(\theta) <$

$e + l\} \subseteq p^k\mathbb{Z}[X]$) we have that $b_e, b_{e+1}, \dots, b_{e+l-1}$ are all divisible by by p^k and so are the leading coefficients of $b_e, b_{e+1}, \dots, b_{e+l-1}$. Hence we get that $d(M') \geq p^{kl}$. Then together with (17) this is a contradiction. $\square$

By the next proposition we are going to have a result to check if $\deg(h_0)$ really is smaller than m, i.e. if $h_0 \in L$. This will be useful in the algorithm to determine the value of m. Clearly one could define $m := n-1$ to be sure that $h_0 \in L$, but we can shorten the running time of our algorithm if we can choose m as small as possible.

Proposition 2.3: Let $b_1, b_2, \dots, b_{m+1}$ be a reduced basis for L and assume $p^{kl} > 2^{mn/2}\binom{2m}{m}^{n/2}|f|^{m+n}$.
Then we have:

$$\deg(h_0) \leq m \iff |b_1| < \left(p^{kl}/|f|^m\right)^{1/n}$$

Proof. We prove both directions:
"$\Leftarrow$": $|b_1| < \left(p^{kl}/|f|^m\right)^{1/n} \Rightarrow p^{kl} > |b_1|^n|f|^m$. Then we have that h_0 divides b_1 in $\mathbb{Z}[X]$ by proposition 2.2 and since $\deg(b_1) \leq m$ we also have that $\deg(h_0) \leq m$.
"$\Rightarrow$": For this part we first need a Theorem of Landau and Mignotte (see [6, page 83]).

Theorem (Landau-Mignotte): Let $f(x) \in \mathbb{Z}[X]$ with degree n and $g(x) \in \mathbb{Z}[X]$ a divisor of $f(x)$ of degree m. Then we have that

$$|g| \leq \binom{2m}{m}^{1/2}|f|$$

(proof see below)
Since $\deg(h_0) \leq m$ we have that $h_0 \in L$ and we can apply the fourth assertion of theorem 1.1 to b_1 and h_0 to get $|b_1| \leq 2^{m/2}|h_0|$. By the fact that h_0 divides f, $\deg(h_0) = l \leq m$ and with Landau-Mignotte we have $|h_0| \leq \binom{2m}{m}^{1/2}|f|$. So we get:

$$\begin{aligned}
|b_1| &\leq 2^{m/2}|h_0| \\
&\leq 2^{m/2}\binom{2m}{m}^{1/2}|f| \\
&= \left(2^{mn/2}\binom{2m}{m}^{n/2}|f|^n\frac{|f|^m}{|f|^m}\right)^{1/n} \\
&< p^{kl/n}/|f|^{m/n} \\
&= \left(p^{kl}/|f|^m\right)^{1/n}
\end{aligned}$$

$\square$

Proof. (of Landau-Mignotte)
For the proof we will need the equation $|(x-a)h| = |a||(x-\overline{a}^{-1})h|$ for $a \in \mathbb{C}$ and $h \in \mathbb{C}[X]$. To see this, let $h := \sum_{i=0}^{n} h_i x^i$ and $h_{-1} = h_{n+1} = 0$ and calculate:

$$\begin{aligned}
|(x-a)h|^2 &= \sum_{i=1}^{n+1} |h_{i-1} - ah_i|^2 \\
&= \sum_{i=0}^{n+1} (h_{i-1} - ah_i)\overline{(h_{i-1} - ah_i)} \\
&= \sum_{i=0}^{n+1} (h_{i-1} - ah_i)(\overline{h_{i-1}} - \overline{h_i}\overline{a}) \\
&= \sum_{i=1}^{n+1} \left(|h_{i-1}|^2 - ah_i\overline{h_{i-1}} - \overline{a}\overline{h_i}h_{i-1} + |ah_i|^2\right) \\
&= \sum_{i=1}^{n+1} \left(|h_i| - ah_i\overline{h_{i-1}} - \overline{a}\overline{h_i}h_{i-1} + |ah_{i-1}|^2\right) \\
&= \sum_{i=0}^{n+1} |\overline{a}h_{i-1} - h_i|^2 \\
&= |(\overline{a}x - 1)h|^2 \\
&= |a|^2|(x - \overline{a}^{-1})h|^2.
\end{aligned}$$

Where in the fifth line we used that $\sum_{i=0}^{n+1} |h_i|^2 = \sum_{i=0}^{n+1} |h_{i-1}|^2 + |h_{n+1}|^2 - |h_{-1}|^2 = \sum_{i=1}^{n+1} h_{i-1}$.
Now let $a_1, \ldots, a_s$ be the set of roots of f inside the unit disk and $a_{s+1}, \ldots, a_n$ the set of roots of f outside the unit disk ordered in decreasing absolute value and f_n be the leading coefficient of f. Then we have:

$$\begin{aligned}
|f|^2 &= |f_n \prod_{i=1}^{s}(x-a_i) \prod_{i=s+1}^{n}(x-a_i)|^2 \\
&= |a_1 a_2 \ldots a_s|^2 |f_n \prod_{i=1}^{s}(x - \overline{a_i}^{-1}) \prod_{i=s+1}^{n}(x-a_i)|^2 \\
&= |a_1 a_2 \ldots a_s|^2 |f_n x^n + \ldots + f_n \prod_{i=1}^{s} \overline{a_i}^{-1} \prod_{i=s+1}^{n} a_i|^2 \\
&\geq |a_1 a_2 \ldots a_s|^2 |f_n \prod_{i=1}^{s} \overline{a_i}^{-1} \prod_{i=s+1}^{n} a_i|^2 \\
&= |f_n \prod_{i=s+1}^{n} a_i|^2
\end{aligned}$$

Let $b_1, \ldots, b_m$ be the roots of $g(x)$ ordered such that $b_i \geq b_{i+1}$ for all $i \in [1, \ldots, m-1]$ and $g(x) = g_m \prod_{i=1}^m (x - b_i) = \sum_{i=0}^m g_i x^i$. Let S_i be the set of all subsets of $b_1, \ldots, b_m$ with $m - i$ elements. Then:

$$|g_i| = |g_m \sum_{S_i} \left(\prod_{b_j \in S_i} b_j \right) |$$

Since there are $\binom{m}{m-i} = \binom{m}{i}$ such subsets in S_i and the absolute value of the product of the elements of such a subset is at most $|b_1 \ldots b_{m-i}|$, we get:

$$|g_i| \leq g_m \binom{m}{i} |b_1 \ldots b_{m-i}|$$

We have that g divides f, so $|b_1 \ldots b_{m-i}| \leq |a_{s+1} \ldots a_{s+m-i}|$ and

$$|g_i| \leq g_m \binom{m}{i} |a_{s+1} \ldots a_{s+m-i}| \leq g_m \binom{m}{i} |a_{s+1} \ldots a_n| \leq \binom{m}{i} \frac{|g_m|}{|f_n|} |f|$$

Furthermore we have that g_m divides f_n because g divides f and therefore $g_m / f_n \leq 1$ and we get

$$|g_i| \leq \binom{m}{i} |f|$$

Finally we have

$$|g| = \left(\sum_{i=0}^m |g_i|^2 \right)^{1/2} \leq \left(\sum_{i=0}^n \binom{m}{i}^2 |f|^2 \right)^{1/2} = \binom{2m}{m}^{1/2} |f|$$

Where the last equality follows from the identity $\sum_{i=0}^m \binom{m}{i}^2 = \binom{2m}{m}$. □

Now we have the final proposition that tells us how to calculate h_0:

Proposition 2.4: As in proposition 2.3, let $b_1, b_2, \ldots, b_{m+1}$ be a reduced basis for L and $p^{kl} > 2^{mn/2} \binom{2m}{m}^{n/2} |f|^{m+n}$. Let t be the greatest integer in $\{1, 2, \ldots, m+1\}$ such that $|b_t| < \left(p^{kl} / |f|^m \right)^{1/n}$.
Then we have:

$$\deg(h_0) = m + 1 - t \text{ and } h_0 = gcd(b_1, b_1, \ldots, b_t).$$

Proof. Let J be the set of all indices j such that $|b_j| < \left(p^{kl} / |f|^m \right)^{1/n}$. With proposition 2.2 we now have that h_0 divides all the b_j for $j \in J$. So we define $h_1 := \gcd(\{b_j; j \in J\})$ and we will show later that $h_0 = h_1$. Clearly h_0 divides h_1.

Moreover h_1 divides all the b_j ($j \in J$) and the degree of b_j is smaller than m. So b_j is an element of the lattice L_1 defined through the basis

$$\{h_1 X^i; 0 \leq i \leq m - \deg(h_1)\}.$$

By definition the b_j's are linearly independent and there are at most $m + 1 - \deg(h_1)$ linearly independent elements in the lattice L_1, so there are at most $m + 1 - \deg(h_1)$ elements in J.
Furthermore we have $h_0 X^i \in L$ for $i \in \{0, \ldots, m - \deg(h_0)\}$ and by the fourth assertion of theorem 1.1 we get:

$$|b_k| \leq 2^{m/2} \max\{|x_i|; 0 \leq i \leq m - \deg(h_0)\} = 2^{m/2}|h_0 X^i| = 2^{m/2}|h_0|$$

By Landau-Mignotte we have $|X^i h_0| \leq \binom{2m}{m}^{1/2}|f|$ for all $i \in \{0, \ldots, m - \deg(h_0)\}$, so we get:

$$|b_k| \leq 2^{m/2}\binom{2m}{m}^{1/2}|f| < \left(p^{kl}/|f|^m\right)^{1/n}$$

for all $k \in \{1, \ldots, m + 1 - \deg(h_0)\}$.
Note that J was defined to be all the indices j such that exactly this inequality holds, so $\{1, \ldots, m + 1 - \deg(h_0)\} \subset J$. Since $\deg(h_0) \leq \deg(h_1)$ and with the observation above about the upper bound for the number of elements in J we get

$$\#\{1, \ldots, m{+}1{-}\deg(h_1)\} = m{+}1{-}\deg(h_1) \leq \#\{1, \ldots, m{+}1{-}\deg(h_0)\} \leq \#J \leq m{+}1{-}\deg(h_1)$$

so we get that $\deg(h_0) = \deg(h_1)$ and $J = \{1, \ldots, m + 1 - \deg(h_0)\}$ and therefore $t := m + 1 - \deg(h_0)$. From that last equality we receive that $\deg(h_0) = m + 1 - t$.
The only thing left to show is that h_0 is indeed h_1. We already know that they have the same degree and that h_0 divides h_1, so we already know that they are equal up to a factor in $\mathbb{Z}$. We claim that this factor equals one, i.e. that h_1 is primitive and therefore its content is one. Then $h_0 = h_1$.
To see that the claim is true, choose some $j \in J$ arbitrary and let c_j be the content of b_j. We know that h_0 divides all b_j and h_0 is primitive, so h_0 also divides b_j/c_j. By the definition of L we then have that $b_j/c_j \in L$. But b_j was defined to be an element of a basis of L, so $c_j = 1$ for all $j \in \{1, \ldots, t\}$ and hence also the content of $h_1 = \gcd(b_1, \ldots, b_t)$ is one, so h_1 is primitive. $\square$

2.2 Determination of the setting

In order to use the above results for the factorization of polynomials, we need to find all polynomials h such that (12), (13), (14) and (15) are true. Obviously also p and k need to be specified. From (15) it follows that h^2 is not allowed to divide f, hence no multiple roots are allowed. For this chapter we therefore assume f has no multiple root and we going to consider the case of multiple roots later.

We first specify the prime p. We will see, that the choice of p will fix (15).
Then, we focus on the factorization of $(f \mod p)$ into irreducible factors in $\mathbb{Z}/p\mathbb{Z}$. The factors we will find will fulfil (12) and (14).
At the end, we set k such that we can modify h (but not $(h \mod p)$) in a way that (13) is fulfilled.

2.2.1 Specification of p

To specify p we first need to calculate the **resultant** of f and its first derivative. The resultant of two polynomials P and Q is defined to be:

$$R(P,Q) := \prod_{(x,y);P(x)=0=Q(y)} (x-y)$$

Note that $R(f,f')$ is only zero if f and f' have one or more common roots, which would then imply that f has multiple roots. Since we defined f to be squarefree $R(f,f') \neq 0$.

We can now define p to be the smallest prime not dividing $R(f,f')$. This is reasonable because of the following arguments:
We know that $R(f,f')$ is up to sign equal to the product of the leading coefficient f_n and the discriminant $D(f)$ of f. So since $R(f,f') \neq (0 \mod p)$, we also have $f_nD(f) \neq (0 \mod p)$ and therefore $f_n \neq (0 \mod p)$ and $D(f) \neq (0 \mod p)$.
We claim that there are no multiple roots in $(f \mod p)$.
We see this if we choose two roots of f arbitrary, say x_i and x_j. Because $(x_i - x_j)$ is a factor of $D(f)$, we have that $(x_i - x_j) \neq (0 \mod p)$ and therefore the difference of x_i and x_j is not a multiple of p. Hence $(x_i \mod p) \neq (x_j \mod p)$ and $((x - x_i) \mod p) \neq ((x - x_j) \mod p)$. But we would need $(x - (x_i \mod p))$ to be $(x - (x_j \mod p))$ for some choice of x_i and x_j to have multiple roots in $(f \mod p)$, which proves that there are none.

As mentioned before, this choice of p ensures that (15) holds for every h with $(h \mod p)$ being a divisor of $(f \mod p)$.

2.2.2 Berlekamp's algorithm

Our goal is now to find the complete factorization of $(f \mod p)$ into irreducible factors in $\mathbb{Z}/p\mathbb{Z}[X]$. Assume that $f(x)$ is already reduced modulo p and that $f(x)$ is square free. Moreover assume that there exists a polynomial $\tilde{f}(x) = \prod_{a \in \mathbb{Z}/p\mathbb{Z}}(g(x) - a) \in \mathbb{Z}/p\mathbb{Z}[X]$ such that $f(x)$ divides $\tilde{f}(x)$. Then every irreducible factor of $f(x)$ also is a irreducible factor of $\tilde{f}(x)$ and we get:

$$f(x) = \gcd(f(x), \tilde{f}(x)) = \gcd\left(f(x), \prod_{a\in\mathbb{Z}/p\mathbb{Z}} (g(x)-a)\right) = \prod_{a\in\mathbb{Z}/p\mathbb{Z}} \gcd(f(x),(g(x)-a))$$

Clearly not every such gcd will be an irreducible factor so we need to find enough (in an appropriate matter) polynomials $g(x)$ to factor out $f(x)$ completely into irreducible factors.
First note that $\prod_{a\in F}(X-a) = X^p - X$ for a finite field F, so for $\tilde{f}(x)$ we get $\prod_{a\in\mathbb{Z}/p\mathbb{Z}}(g(x)-a) = g(x)^p - g(x)$. Since $\tilde{f}(x)$ is divisible by $f(x)$ we get that $(g(x)^p - g(x))$ is divisible by $f(x)$ and therefore:

$$g(x)^p = g(x) \mod f(x). \tag{18}$$

So we can restrict the search of $g(x)$ to the set with this property. This set is also called the Berlekamp subalgebra and it has some nice property that will help us find $g(x)$.
We now define the matrix $Q = \{q_{kl}\}_{0\le k,l\le n\}}$ with entries q_{kl} given by the equation

$$x^{ip} = \left(q_{n,i}x^n + q_{n-1,i}x^{n-1} + \ldots + q_{0,i}\right) \mod f(x) \text{ for } i \in [0,\ldots,n]$$

Claim: $g(x)$ fulfills (18) if and only if $g(x)$ is a eigenvector of Q with eigenvalue one.

Proof. Note first $(X+Y)^p = X^p + Y^p$ in $\mathbb{Z}/p\mathbb{Z}$ as a consequence of the binomial theorem. Moreover $b^p = b$ for $b \in \mathbb{Z}/p\mathbb{Z}$. We see this with the equation $\prod_{a\in\mathbb{Z}/p\mathbb{Z}}(b-a) = (b^p - b)$. Then obviously if $b \in \mathbb{Z}/p\mathbb{Z}$ the left hand side is zero and so is the right hand side when reduced modulo p, hence $(b^p - b) = 0$ in $\mathbb{Z}/p\mathbb{Z}$.
Assume now that $g(x)$ is an eigenvector of Q, i.e if $g(x) = g_n x^n + g_{n-1}x^{n-1} + \ldots + g_0$, then:

$$\begin{aligned}
g(x) &= \sum_{i=0}^{n} g_i x^i = \sum_{i=0}^{n}\left(\sum_{j=0}^{n} q_{ij}b_j\right)x^i = \sum_{j=0}^{n} b_j \sum_{i=0}^{n} q_{ij}x^j \\
\iff g(x) &= \sum_{j=1}^{n} b_j(x^{jp} \mod f(x)) = \sum_{j=0}^{n}(b_j^p x^{jp}) \mod f(x) \\
&= \left(\sum_{j=0}^{n} b_j x^j\right)^p \mod f(x) = g(x)^p \mod f(x)
\end{aligned}$$

□

With this result we are now able to describe an algorithm that factors the polynomial $f(x)$ in $\mathbb{Z}/p\mathbb{Z}[X]$:

- First we need to calculate Q. This can be done by calculating $x^{ip} \mod f(x)$ for all $i \in [0, 1, \dots, n]$.
- Then we need to calculate all the eigenvectors $g_j(x)$ for $j \in [1, \dots, \operatorname{rank}(Q - Id)]$. (Since there are exactly $\operatorname{rank}(Q - Id)$ linearly independent vectors satisfying the equation $(Q - Id)g = 0$)
- For all $g_j(x)$ and all $a \in \mathbb{Z}/p\mathbb{Z}$ we now have to calculate $\gcd(f(x), (g_j(x) - a))$, which can be done by the Euclidean Algorithm.
  Note that as soon as the algorithm finds $\operatorname{rank}(Q - Id)$ different factors, it can stop.
- Repeat this procedure for all factors of $f(x)$ we found so far, untill all factors are irreducible.

2.2.3 Hensel's Lift

It remains to specify the integer k. We need to specify k for every factor of $(h \mod p)$ seperately. So we choose one of them and call it $(h \mod p)$.
First note that if we set $k = 1$, (12), (13), (14) and (15) are true for the prime p we specified earlier and for the factor $(h \mod p)$ calculated by the Berlekamp's Algorithm. In order to be able to use the results form the beginning, we need the equation $p^{kl} > 2^{mn/2}\binom{2m}{m}^{n/2}|f|^{m+n}$ (see Proposition 2.3) to hold. Recall that m has to be greater than $\deg(h_0)$, otherwise h_0 is not in the lattice we defined and we cannot find h_0 with the results above. Since in the worst case $\deg(h_0) = n - 1$ we set $m := n - 1$ in the above inequality and define k to be the least integer such that the inequality holds, i.e:

$$k := \min\{k \in \mathbb{Z}; p^{kl} > 2^{n(n-1)/2}\binom{2n-2}{n-1}^{n/2}|f|^{2n-1}\}$$

Now (13) is not necessary true, therefore we need to modify h such that $(h \mod p)$ does not change but (13) becomes true. This modification can be performed by Hensel's lift.

Theorem 2.5 (Hensel's Lemma): Let f be a monic polynomial in $\mathbb{Z}[X]$ and $(h \mod p^{i-1})$ a irreducible factor of $(f \mod p^{i-1})$ for an integer $i \geq 2$. Then there exists a polynomial $\tilde{h}$ (uniquely up to $\mod p^i$) such that $(\tilde{h} \mod p^i)$ divides $(f \mod p^i)$ and $(\tilde{h} \mod p^{i-1}) = (h \mod p^{i-1})$.

Proof. Since $(h \mod p^{i-1})$ divides $(f \mod p^{i-1})$ we have that there exists an polynomial $g(x) \in \mathbb{Z}[X]$ such that $(f \mod p^{i-1} = (h \mod p^{i-1})(g \mod p^{i-1})$. Moreover $(\tilde{h} \mod p^{i-1})$ has to be equal to $(h \mod p^{i-1})$, so $\tilde{h} = h + up^{i-1}$ for some $u \in \mathbb{Z}[X]$ with

$\deg(u) \leq \deg(h)$.
Since $(\tilde{h} \mod p^i)$ has to divide $(f \mod p^i)$ we have that there existst a $\tilde{g} \in \mathbb{Z}[X]$ such that $(f \mod p^i) = (\tilde{h} \mod p^i)(\tilde{g} \mod p^i)$. Set $\tilde{g} = g + vp^{i+1}$ for an appropriate choice of $v \in \mathbb{Z}[X]$ with $\deg(v) \leq \deg(g)$. Then we have:

$$\begin{aligned}
&f \mod p^i = (h + up^{i-1})(g + vp^{i-1}) \mod p^i \\
\Rightarrow &f \mod p^i = (hg + (ug + vh)p^{i-1} + uvp^{2i-2}) \mod p^i \\
\Rightarrow &f/p^{i-1} \mod p^i = (hg/p^{i-1} + (ug + vh)) \mod p^i \\
\Rightarrow &\left(\frac{f - hg}{p^{i-1}}\right) \mod p^i = (ug + vh) \mod p^i
\end{aligned}$$

We had $f = hg \mod p^{i-1}$, so $f = hg + cp^{i-1}$ where $c \in \mathbb{Z}$, therefore $f - hg = cp^{i-1}$ and $(f - hg)/p^{i-1} = c$. Then:

$$\begin{aligned}
&(c \mod p^i) \mod p = \big((ug + vh) \mod p^i\big) \mod p \\
\Rightarrow &c \mod p = (ug + vh) \mod p
\end{aligned}$$

and hence there exists unique u and v so also $\tilde{h}$ and $\tilde{g}$. $\square$

Now we can apply Hensel's Lift repeatedly for $i = 2, \ldots, k$ to get the desired modification of $h \mod p$.

2.3 The algorithm

We are now able to describe an algorithm that factors a given polynomial $f(x) \in \mathbb{Z}[X]$ into irreducible factors in $\mathbb{Z}[X]$.
For the algorithm we need $f(x)$ to be primitive, but note that if it is not, then we can easily calculate the greatest common divisor of its coefficients and take $f_0(x)$ to be $f(x)$ divided through the greatest common divisor to get a primitive polynomial.
As already mentioned we also need polynomials with no multiple factors, so this is the first thing the algorithm has to check for. We can do this with the calculation of the resultant of f and its first derivative. As already said, f has multiple roots if and only if $R(f, f')$ is zero.
Assume $R(f, f')$ is indeed zero. then we will calculate $g := \gcd(f, f')$, which is then the set of multiple factors of f. Set $f_0(x) := f(x)/g(x)$, then obviously $f_0(x)$ has no multiple factors. After factoring $f_0(x)$ it will then be easy to find the factorization of $g(x)$ since $g(x)$ only has factors that appear in the factorization of $f_0(x)$ and there are only finitely many factors in $f_0(x)$.
Now that we have a polynomial with no multiple factors, we determine the prime number p and the set $F := \{h \mod p; (h \mod p) \text{ is a irreducible factor of } (f \mod p)\}$ with

Berlekamp's algorithm.
In order to factor out $f(x)$ we now assume in each step that $f = f_1 f_2$ where f_1 is already factored out in irreducible factors and we let F_2 be the set of all irreducible factors of $f_2 \mod p$. Then we choose one of the elements of F_2 (say h), calculate k as above and modify h as in Hensel's Lift.
We can now calculate h_0 with the result from Proposition 2.4. First we need to define m. For this we just take $m := n-1$ because then we secured that h_0 is an element of the lattice. So we obtain a reduced basis for the lattice $\{p^k X^i; 0 \leq i < l\} \cup \{hX^j; 0 \leq j \leq m-l\}$ by the use of the LLL-Algorithm, and then calculate h_0 as in proposition 24.
We now set $\tilde{f_1} := f_1 h_0$ and $\tilde{f_2} := f_2/h_0$ and repeat the whole procedure for $\tilde{f_2}$ until we get $f_2 \equiv 1$. Note that we only have to calculate the irreducible factors of $f_2 \mod p$ once, and not for every single loop. We can just delete all the factors that divide $h_0 \mod p$.

Note that we can choose m smaller such that the running time of the algorithm becomes shorter. This works the following way: Let u be such that $l \leq \lfloor \frac{n-1}{2^u} \rfloor$. Choose $m_i = \lfloor \frac{n-1}{2^{u-i}} \rfloor$ for $0 \leq i \leq u$ and check if $\deg(h_0) \leq m_i$ by the result of proposition 2.3 for every value m_i. As soon as $\deg(h_0) \leq m_j$ for some j, calculate h_0 as in proposition 2.4. So for every choice of m_i we first determine a reduced basis $b_1, b_2, \ldots, b_{m+1}$ by the LLL-Algorithm and then check if $\deg(h_0) \leq m_i$ by checking if $|b_1| < \left(p^{kl}/|f|^m\right)^{1/n}$. If indeed the inequality holds, we can calculate h_0 by the equation $h_0 = \gcd(b_1, \ldots, b_j)$, if not, continue with the next value m_{j+1}. Since m goes up to $n-1$ it is guaranteed that we will find h_0 sooner or later. (if not then $h_0 = f$)

Algorithm:

```
1. $r := R(f, f')$;
2. $g := 1$;
3. **if** $r = 0$
4.     $g := \gcd(f, f')$; $f := f/g$; $r := R(f, f')$;
5. **end**
6. $p :=$ smallest prime number such that $p$ does not divide $R(f, f')$;
7. $F := \{h \mod p; (h \mod p)$ irreducible factor of $(f \mod p)\}$ (by Berlekamp's Algorithm)
```

```
8.  $f_1 := 1$; $f_2 := f$:
9.  **while** $f_2 \neq 1$
10.     $h :=$ some arbitrary element of $F$;
11.     $l := \deg(h)$;
12.     **if** $l = n$, $h_0 = f$; **break**
13.     $k := \min\{k \in \mathbb{N}; p^{kl} > n^{(n-1)n/2}\binom{2n-2}{n-1}^{n/2}|f|^{2n-1}\}$;
14.     Modify $h$ by means of Hensel's Lift;
15.     $m := n - 1$;
16.     Apply LLL-Algorithm to $\{p^k X^i; 0 \leq i < l\} \cup \{hX^j; 0 \leq j \leq m - l\}$
17.     $j :=$ greatest integer such that $|b_j| < \left(p^{kl}/|f|^m\right)^{1/n}$ (see proposition 2.4)
18.     $h_0 := \gcd(b_1, \ldots, b_j)$
19.     $f_1 := f_1 h_0$; $f_2 := f_2/h_0$;
20.     $H_0 := \{h \mod p; (h \mod p) \text{ divides } (h_0 \mod p)\}$;
21.     $F := F \setminus H_0$;
22. **end**
23. $g_1 = 1$; $g_2 := g$;
24. **while** $g_2 \neq 1$
25.     Check all factors of $f$ if they divide $g_2$. Let $\tilde{f}$ divide $g_2$.
26.     $g_1 := g_1\tilde{f}$; $g_2 := g_2/\tilde{f}$;
27. **end**
```

If we use the observation above that we can optimize the running time of the algorithm by some smaller value of m, we have to replace line 16 to 19 by the following algorithm:

```
1. $u := \max\{u \in \mathbb{N}; l < (n-1)/2^u\}$;
2. while $m \leq (n-1)$
3.     $m := \lfloor (n-1)/2^u \lfloor$;
4.     Apply LLL-Algorithm to $\{p^k X^i; 0 \leq i < l\} \cup \{hX^j; 0 \leq j \leq m-l\}$
5.     if $|b_1| < \left(p^{kl}/|f|\right)^{1/n}$
6.         $h_0 := \gcd(b_1, \ldots, b_j)$;
7.         $m := n$;
8.     end
9.     $u := u - 1$;
10. end
```

3 Integer relation

Another application of the LLL-Algorithm is to search for integer relations.

Definition 3.1: Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be real numbers. Then the vector $m = (m_1, m_2, \ldots, m_n)^T \in \mathbb{Z}^n$ is called an integer relation for $\alpha_1, \alpha_2, \ldots, \alpha_n$ if $\sum_{i=1}^n m_i \alpha_i = 0$.

Our goal is now to find integer relation to a given set of real numbers or show that there are no integer relation.
In order to apply the LLL-Algorithm we clearly need to define a lattice first.
Since a computer cannot calculate with real numbers we need to approximate $\alpha_1, \alpha_2, \ldots, \alpha_n$ by rationals $\overline{\alpha_1}, \ldots, \overline{\alpha_n} \in \mathbb{Q}$. This approximation requires to fulfill certain properties to really get useful results with the reduction algorithm.
We define a basis of the lattice to be the vectors:

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ c\overline{\alpha_1} \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ c\overline{\alpha_2} \end{pmatrix}, \ldots, v_n = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ c\overline{\alpha_n} \end{pmatrix}$$

where c is an appropriate large constant.
Then we apply the reduction algorithm to these vectors. Now b_1 is of the form $m_1v_1 + \ldots + m_nv_n$ with $m_1, \ldots, m_n$ being integers.
By theorem 1.1 (fourth assertion) we have $|b_1| \leq 2^{(n-1)/2}|x|$ for every $x \in L$, so

$$\begin{aligned}|b_1|^2 &= |m_1^2 + m_2^2 + \ldots + m_n^2 + (c\sum_{i=1}^{n} m_i\overline{\alpha_i})^2| \\ &= |m|^2 + c^2(\sum_{i=1}^{n} m_i\overline{\alpha_i})^2 \\ &\leq 2^{n-1}|x|^2\end{aligned}$$

hence we have an upper bound for $|b_1|^2$. We can choose c and $\alpha_1, \alpha_2, \ldots, \alpha_n$ in such a way, that if we have that $|b_1|$ is below a certain value we know that the last entry of b_1 ($\sum_{i=1}^{n} m_i\overline{\alpha_i}$) is so small that we can conclude it is zero and hence $m_1, \ldots, m_n$ a integer relation. On the other hand also the converse is true, i.e. if $|b_1|$ is not below a certain bound then m is no integer relation for $\alpha_1, \alpha_2, \ldots, \alpha_n$ (for the exact bounds and the choice of c and $\alpha_1, \alpha_2, \ldots, \alpha_n$ see [7]).

4 BBP-Type formulae

Peter Borwein and Simon Plouffe observed in 1995, that with the well-known identity $\log 2 = \sum_{k=1}^{\infty} \frac{1}{k2^k}$ one can calculate an arbitrary digit of $\log 2$ in base 2 without knowing the proceeding digits. So they started searching for other mathematical constants with the same properties.
This search basically is a search for integer relations. They performed this task by using the **PSLQ-Algorithm** written by David H. Bailey and Helaman Ferguson in 1992. The PSLQ-Algorithm is not a lattice-reduction algorithm, but one can perform integer relation search faster than with the LLL-Algorithm.
In 1997, Bailey, Borwein and Plouffe finally introduced an algorithm to compute the d'th hexadecimal digit of π using a identity for π they found by the PSLQ-Algorithm. The fact that the digits of π were normal misled people into disbelieving the existence of such an algorithm. In that sense, this result was pioneering, since now we have a formula to calculate these digits without knowing the preceeding ones.
Although the new algorithm is not really faster than the preceeding algorithms calculating all the digits up to the desired one, it's still very useful since one can execute it on a normal personal computer, and its implementation is easier than that of all the preceeding algorithms.
However, we don't know yet whether there is a series for π or $\log 2$ in base 10 or some power of 10 or any other base than 16 or a power of 2 respectively. Calculations done so far tell us that there are no such formulas for coefficients in certain ranges, but still there is the possibility that such formulas exist.

In this chapter we will first show the **formula for** π in hexadecimal basis and a proof for that formula. Then some other **BBP-Type formulae** and finally the algorithm to calculate the d'th digit of an arbitrary BBP-Type formula.

4.1 A formula for π

Claim: The following identity holds:

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right) \tag{19}$$

Note: There are several other formulas for π of this type. But since the formula written above was the first one to be announced and hence the most popular one, we are just going to prove this one. The proofs of the others are completely similar.
Moreover we have the equation:

$$0 = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(-\frac{8}{8i+1} + \frac{8}{8i+2} + \frac{4}{8i+3} + \frac{8}{8i+4} + \frac{2}{8i+5} + \frac{2}{8i+6} - \frac{1}{8i+7} \right) \tag{20}$$

and it turned out that all the known formulas for π can be written as formula (19) plus a multiple of (20).

Preparation for the proof: First note:

$$\frac{x^{k-1}}{1-x^8} = x^{k-1} \sum_{i=0}^{\infty} x^{8i} = \sum_{i=0}^{\infty} x^{k-1+8i} \text{ for } 0 \le x < 1$$

(Geometric series)

hence

$$\begin{aligned}
\int_0^{1/\sqrt{2}} \frac{x^{k-1}}{1-x^8}\mathrm{d}x &= \int_0^{1/\sqrt{2}} \sum_{i=0}^{\infty} x^{k-1+8i}\mathrm{d}x \\
&= \int_0^{1/\sqrt{2}} \lim_{n\to\infty} \sum_{i+1}^{n} x^{k-1+8i}\mathrm{d}x \\
&= \lim_{n\to\infty} \int_0^{1/\sqrt{2}} \sum_{i=0}^{n} x^{k-1+8i}\mathrm{d}x \\
&= \lim_{n\to\infty} \sum_{i=0}^{n} \int_0^{1/\sqrt{2}} x^{k-1+8i}\mathrm{d}x \\
&= \sum_{i=0}^{\infty} \int_0^{1/\sqrt{2}} x^{k-1+8i}\mathrm{d}x \\
&= \sum_{i=0}^{\infty} \left(\frac{1}{8i+k}x^{8i+k}\right)\Big|_0^{1/\sqrt{2}} \\
&= \frac{1}{\sqrt{2}^k} \sum_{i=0}^{\infty} \frac{1}{16^i(8i+k)}.
\end{aligned}$$

We can interchange integral and limit in the third equality because of the monotone convergence theorem.

Proof. From the calculation above,we have the following equality:

$$\begin{aligned}
(19) &= \int_0^{1/\sqrt{2}} \frac{4\sqrt{2}-8x^3-4\sqrt{2}x^4-8x^5}{1-x^8}\mathrm{d}x \\
\overset{y:=\sqrt{2}x}{=} & \int_0^1 \frac{4-2y^3-y^4-y^5}{1-\frac{y^8}{16}}\mathrm{d}y \\
&= \int_0^1 \frac{16y-16}{y^4-2y^3+4y-4}\mathrm{d}y \qquad (21)
\end{aligned}$$

Where in the second equality we used the substitution $y := \sqrt{2}x$ and in the third equality we cancelled the common factor $(y^4+2y^3+4y^2+4y+4)$ in nominator and denominator of the fraction in the integral.
It is now easy to check with some Computer Algebra System (like Maple), that the integral (21) equals π. $\square$

4.2 BBP-Type formulae in general

There were other such sums discovered being equal to some transcendental constant other than π.
In general we are searching for sums of the type

$$\alpha = \sum_{i=1}^{\infty} \frac{p(k)}{b^k q(k)} \tag{22}$$

where α is a constant and p and q are polynomials ($\deg(p) < \deg(q)$) with integer coefficients . The numerical basis b is a positive integer. Formulae of this type are called **BBP-type formulae**. However there is no general algorithm to find a certain combination of p, q and b such that the sum equals some constant. These combinations are currently discovered via a combination of guessing and searching with the PSLQ integer relation algorithm.

Some examples of BBP-type formulae for other constants:

1. The simplest formulae of this type were well-known even before BBP

$$\log(2) = \sum_{i=1}^{\infty} \frac{1}{2^k k}$$

$$\log(\frac{9}{10}) = -\sum_{i=1}^{\infty} \frac{1}{10^k k}$$

   Both of them can easily be checked by expanding $\log(1+1)$ and $\log(1-\frac{1}{10})$ as Taylor series.

2. Less obvious are the identities:

$$\pi^2 = \frac{9}{8}\sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{16}{(6i+1)^2} - \frac{24}{(6i+2)^2} - \frac{8}{(6i+3)^2} - \frac{6}{(6i+4)^2} + \frac{1}{(6i+5)^5}\right)$$

$$\log^2(2) = \frac{1}{8}\sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{-16}{(6i)^2} + \frac{16}{(6i+1)^2} - \frac{40}{(6i+2)^2} - \frac{14}{(6i+3)^2} - \frac{10}{(6i+4)^2} + \frac{1}{(6i+5)^2}\right)$$

4.3 Computing the i-th digit

In the next section, we are going to show how one can actually compute the i-th digit of a constant with a BBP-Type formulae. We will start with the an easy case, $\log 2$.

In all of these computations one has to compute a term of the form $r = b^n \bmod k$. So we will first describe an algorithm to do this calculation. Basically, we use the binary expansion of n, and that one can compute b^n very fast by successive squaring and multiplication.

```
1.  $t := 2^i$ for $i \in \mathbb{N}$ such that $t \leq n \leq 2t$
2.  $r := b \bmod k$
3.  $n := n - t$
4.  $t := t/2$
5.  **while** $t \geq 1$
6.    $r := r^2 \bmod k$
7.    **if** $n \geq t$
8.      $r := br \bmod k$
9.      $n := n - t$
10.   **end**
11.   $t := t/2$
12. **end**
```

With this algorithm one can calculate the expression $r = b^n \bmod k$ very efficient on a computer. Moreover this algorithm works on a normal personal computer since all the numbers the algorithm calculates do not exceed k^2 in size.

4.3.1 Computing binary digits of $\log 2$

As we already know: $\log 2 = \sum_{k=1}^{\infty} \frac{1}{k2^k}$
One can calculate the $(d+1)$'th digit in base 2 as follows:
We first calculate $2^d \log 2$ such that the $(d+1)$'th digit is now at the first position of the decimal part. Note that $(2^d \log 2) \bmod 1$ is the fractional part of $2^d \log 2$.

We have:

$$\begin{aligned}(2^d \log 2) \bmod 1 &= \sum_{k=1}^{\infty} \frac{2^d}{k2^k} \bmod 1 \\ &= \left(\sum_{k=1}^{d} \frac{2^{d-k}}{k} \bmod 1 + \sum_{k=d+1}^{\infty} \frac{1}{2^{k-d}k}\right) \bmod 1 \\ &= \left(\left(\sum_{k=1}^{d} \frac{2^{d-k} \bmod k}{k}\right) \bmod 1 + \sum_{k=d+1}^{\infty} \frac{1}{2^{k-d}k}\right) \bmod 1\end{aligned}$$

Where only the first summation is greater than one (in decimal system) and the "mod k" is justified because we are only interested in the fractional part.
The first summation consists of d terms, and each of them can be calculated on a ordinary personal computer with the algorithm above and floating point arithmetic for the division. For the second summation we only need a few terms to be evaluated since they quickly become sufficiently small.
By this procedure one only has to look at the first digit of the number the computer computed and we have the desired digit.
Note: The result we get by that procedure is a decimal number. One can calculate the binary expansion of this in the following way:
The result of the algorithm is a number of the form $0.n_1n_2n_3...$ where $n_i = 0, 1, ..., 9$. The first digit of that number in the base 2 is the integer part of $2*0.n_1n_2n_3...$ the second number is the integer part of $2*((2*0.n_1n_2n_3...)\text{mod } 1)$ and the j'th number is the integer part of $2*((2^{j-1}*0.n_1n_2n_3...)\text{mod } 1)$. Clearly at some point the number is going to be incorrect since the computer can only calculate decimal numbers up to a certain accuracy as soon as the number gets too small. But the first few digits of the calculation are always correct and since we can repeat this procedure for the $d+2$'th digit, it is not even necessary that the computer calculates more than one digit correct.

4.3.2 Computing hexadecimal digits of π

We now apply the same procedure to the formula of π.
We have:

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right), \tag{23}$$

and by multiplying with 16^d and consider only the fractional part, we get:

$$(16^d\pi) \bmod 1 = (4(16^dS_1 \bmod 1) - 2(16^dS_4 \bmod 1) - (16^dS_5 \bmod 1) - (16^dS_6 \bmod 1))\bmod 1 \tag{24}$$

with

$$S_i = \sum_{k=0}^{\infty} \frac{1}{16^k(8k+i)}$$

and

$$\begin{aligned}(16^d S_i) \mod 1 &= \left(\sum_{k=0}^{d} \frac{16^{d-k}}{8k+i} \mod 1 + \sum_{k=d+1}^{\infty} \frac{1}{16^{k-d}(8k+i)}\right) \mod 1 \\ &= \left(\left(\sum_{k=0}^{d} \frac{16^{d-k} \mod (8k+i)}{8k+i}\right) \mod 1 + \sum_{k=d+1}^{\infty} \frac{1}{16^{k-d}(8k+i)}\right) \mod 1\end{aligned}$$

One does this for all $(16^d S_i)\text{mod } 1, i = 1, 4, 5, 6$ and combine these as in (5). Then add or subtract integers such that the result is between 0 and 1.
Again, we get a decimal number. To translate that into a hexadecimal we use the same strategy as in the case of the binary, i.e. we repeatedly multiply by 16, omit the fractional part and then continue with $(16 * 0.n_1 n_2 n_3 ...)\text{mod } 1$.

4.3.3 The general case

Consider we have a constant defined by a series of the form:

$$S = \sum_{k=0}^{\infty} \frac{1}{b^k q(k)}$$

where b is a positive constant also called the base and $q(k)$ a polynomial with integer coefficients. Again the $(d+1)$'th digit in base b expansion can be obtained by looking at the fractional part of $b^d S$.

$$\begin{aligned}b^d S \mod 1 &= \sum_{k=0}^{\infty} \frac{b^{d-k}}{q(k)} \mod 1 \\ &= \left(\sum_{k=0}^{d} \frac{b^{d-k} \mod q(k)}{q(k)}\right) \mod 1 + \left(\sum_{k=d+1}^{\infty} \frac{1}{b^{k-d} q(k)}\right) \mod 1\end{aligned}$$

Which can be calculated with the above algorithm and floating-point arithmetic on a common personal computer.

References

[1] S. Aland, *Einzelne Ziffern von Pi nach Bailey, Borwein & Plouffe*, (2005), available at `http://math-www.uni-paderborn.de/~aggathen/vorl/2004ws/sem/sebastian-aland.pdf`

[2] D. Bailey, *The BBP Algorithm for Pi*, (2006), available at `http://crd.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf`

[3] D. Bailey, P. Borwein, S. Plouffe *On the Rapid Computation of Various Polylogarithmic Constants*, Mathematics of Computation **66**, pp. 903-913 (1992)

[4] E.R. Berlekamp, *Factoring Polynomials Over Finite Fields*, Bell Systems Technical Journal **46**, pp. 1853-1859 (1967)

[5] J.W.S. Cassels, *An Intorduction to the Geometry of Numbers*, Springer (1971)

[6] A.M.Cohen, H. Cuypers, H. Sterk (Eds.), *Some Tapas of Computer Algebra*, Springer pp. 66-90 (1999)

[7] B. Just, *Integer Relations Among Algebraic Numbers*, Lecture Notes in Computer Sience **379**, pp. 314-320 (1989)

[8] A.K. Lenstra, H.W. Lenstra Jr., L. Lovasz, *Factoring Polynomials with Rational Coefficients*, Mathematische Annalen **261**, pp. 515-534 (1982)

Lecture 3
LLL, Coppersmith

Instructor: Chris Peikert
Scribe: Yan Wang

1 The LLL Algorithm

Recall the definition of an LLL-reduced lattice basis.

Definition 1.1. A lattice basis $\mathbf{B}$ is *LLL-reduced* if the following two conditions are met:

1. For every $i < j$, we have $|\mu_{i,j}| \leq \frac{1}{2}$. (Such a basis is said to be "sized reduced.")
2. For every $1 \leq i < n$, we have $\frac{3}{4}\|\widetilde{\mathbf{b}}_i\|^2 \leq \|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|^2$. (This is the "Lovász condition.")

The LLL algorithm works as follows: given an *integral* input basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ (the integrality condition is without loss of generality), do the following:

1. Compute $\widetilde{\mathbf{B}}$, the Gram-Schmidt orthogonalized vectors of $\mathbf{B}$.
2. Let $\mathbf{B} \leftarrow \mathsf{SizeReduce}(\mathbf{B})$.

   (This algorithm, defined below, ensures that the basis is size reduced, and does not change $\mathcal{L}(\mathbf{B})$ or $\widetilde{\mathbf{B}}$.)
3. If there exists $1 \leq i < n$ for which the Lovász condition is violated, i.e., $\frac{3}{4}\|\widetilde{\mathbf{b}}_i\|^2 > \|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|^2$, then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ and go back to Step 1. Otherwise, output $\mathbf{B}$.

The idea behind the $\mathsf{SizeReduce}(\mathbf{B})$ subroutine is, in the Gram-Schmidt decomposition $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$, to shift the entries in the upper triangle of $\mathbf{U}$ by integers (via unimodular transformations), so that they lie in $[-\frac{1}{2}, \frac{1}{2})$. Because changing an entry of $\mathbf{U}$ may affect the ones above it (but not below it) in the same column, we must make the changes upward in each column. Formally, the algorithm works as follows:

- For each $j = 2, \ldots, n$ (in any order) and $i = j - 1$ *down to* 1, let $\mathbf{b}_j \leftarrow \mathbf{b}_j - \lfloor \mu_{i,j} \rceil \cdot \mathbf{b}_i$, where $\mu_{i,j} = \langle \mathbf{b}_j, \widetilde{\mathbf{b}}_i \rangle / \langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle$ is the (i, j)th entry of the upper-unitriangular matrix in the Gram-Schmidt decomposition of the *current* basis $\mathbf{B}$. (Note that previous iterations can change this matrix.)

  In matrix form, in the (i, j)th iteration we are letting $\mathbf{B} \leftarrow \mathbf{B} \cdot \mathbf{W}$, where $\mathbf{W}$ is the upper unitriangular matrix with just one potentially nonzero off-diagonal entry $-\lfloor \mu_{i,j} \rceil$, at position (i, j).

We make a few important observations about the SizeReduce algorithm. First, it clearly runs in time polynomial in the bit length of the input basis $\mathbf{B}$. Second, even though $\mathbf{B}$ may change, the Gram-Schmidt vectors $\widetilde{\mathbf{B}}$ are preserved throughout, because the only changes to $\mathbf{B}$ are via multiplication by upper-unitriangular matrices, i.e., if $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$ is the Gram-Schmidt decomposition prior to some iteration, then $\mathbf{B} = \widetilde{\mathbf{B}} \cdot (\mathbf{U}\mathbf{W})$ is the decomposition afterward, since $\mathbf{U}\mathbf{W}$ is upper unitriangular. Finally, the (i, j)th iteration ensures that the value $\langle \mathbf{b}_j, \widetilde{\mathbf{b}}_i \rangle / \langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle \in [-\frac{1}{2}, \frac{1}{2})$ (by definition of $\mu_{i,j}$), and following the iteration, that value never changes, because $\mathbf{b}_k$ is orthogonal to $\widetilde{\mathbf{b}}_i$ for all $k < i$. (This is why it important that we loop from $i = j - 1$ *down to* 1; $\mathbf{b}_k$ may not be orthogonal to $\widetilde{\mathbf{b}}_i$ for $k > i$.) Putting these observation together, we have the following lemma on the correctness of SizeReduce.

Lemma 1.2. *Given an integral basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ with Gram-Schmidt decomposition $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$, the* SizeReduce *algorithm outputs a basis $\mathbf{B}'$ of $\mathcal{L} = \mathcal{L}(\mathbf{B})$ having Gram-Schmidt decomposition $\mathbf{B}' = \widetilde{\mathbf{B}} \cdot \mathbf{U}'$, where every entry $u'_{i,j}$ for $i < j$ is in $[-\frac{1}{2}, \frac{1}{2})$.*

We now state the main theorem about the LLL algorithm.

Theorem 1.3. *Given an integral basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, the LLL algorithm outputs an LLL-reduced basis of $\mathcal{L} = \mathcal{L}(\mathbf{B})$ in time $\mathrm{poly}(n, |\mathbf{B}|)$, where $|\mathbf{B}|$ denotes the bit length of the input basis.*

The remainder of this section is dedicated to an (almost complete) proof of this theorem. First, it is clear that the LLL algorithm, if it ever terminates, is correct: all the operations on the input basis preserve the lattice it generates, and the algorithms terminates only when the basis is LLL-reduced.

We next prove that the number of iterations is $O(N)$ for some $N = \mathrm{poly}(n, |\mathbf{B}|)$. This uses a clever "potential argument," which assigns a value to all the intermediate bases produced by the algorithm. We show three facts: that the potential starts out no larger than 2^N, that it never drops below 1, and that each iteration of the algorithm decreases the potential by a factor of at least $\sqrt{4/3} > 1$. This implies that the number of iterations is at most $\log_{\sqrt{4/3}} 2^N = O(N)$.

The potential function is defined as follows: for a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, let $\mathcal{L}_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ for each $1 \leq i \leq n$. The potential is the product of these lattices' determinants:

$$\Phi(\mathbf{B}) := \prod_{i=1}^{n} \det(\mathcal{L}_i) = \prod_{i=1}^{n} \Big(\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_i\| \Big) = \prod_{i=1}^{n} \|\widetilde{\mathbf{b}}_i\|^{n-i+1}.$$

Claim 1.4. *The potential of the initial input basis* $\mathbf{B}$ *is at most* 2^N *where* $N = \mathrm{poly}(n, |\mathbf{B}|)$*, and every intermediate basis the algorithm produces has potential at least* 1.

Proof. The potential of the original basis $\mathbf{B}$ is clearly bounded by $\prod_{i=1}^{n} \|\mathbf{b}_i\|^n \leq \max_i \|\mathbf{b}_i\|^{n^2} = 2^{\mathrm{poly}(n,|\mathbf{B}|)}$. Every intermediate basis is integral and has positive integer determinant, hence so do the lattices $\mathcal{L}_i$ associated with that basis. Therefore, the potential of that basis is at least 1. □

We next analyze how the potential changes when we perform a swap in Step 3.

Claim 1.5. *Suppose* $\mathbf{b}_i$ *and* $\mathbf{b}_{i+1}$ *are swapped in Step 3, and let the resulting basis be denoted* $\mathbf{B}'$*. Then* $\widetilde{\mathbf{b}}'_j = \widetilde{\mathbf{b}}_j$ *for all* $j \notin \{i, i+1\}$*, and* $\widetilde{\mathbf{b}}'_i = \mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}$.

Proof. For $j < i$, the vector $\widetilde{\mathbf{b}}'_j$ is unaffected by the swap, because by definition it is the component of $\mathbf{b}'_j = \mathbf{b}_j$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{j-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1})$. Similarly, for $j > i+1$, the vector $\widetilde{\mathbf{b}}'_j$ is the component of $\mathbf{b}'_j = \mathbf{b}_j$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{j-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1})$, where the equality holds because both $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ are in the span. Finally, $\widetilde{\mathbf{b}}'_i$ is the component of $\mathbf{b}'_i = \mathbf{b}_{i+1}$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$, which is $\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}$ by construction. □

Lemma 1.6. *Suppose* $\mathbf{b}_i$ *and* $\mathbf{b}_{i+1}$ *are swapped in Step 3, and let the resulting basis be denoted* $\mathbf{B}'$*. Then* $\Phi(\mathbf{B}')/\Phi(\mathbf{B}) < \sqrt{3/4}$.

Proof. Let $\mathcal{L}_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_i)$ and $\mathcal{L}'_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1})$. By Claim 1.5, we have

$$\frac{\Phi(\mathbf{B}')}{\Phi(\mathbf{B})} = \frac{\det(\mathcal{L}'_i)}{\det(\mathcal{L}_i)} = \frac{\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_{i-1}\| \|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|}{\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_{i-1}\| \|\widetilde{\mathbf{b}}_i\|} = \frac{\|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|}{\|\widetilde{\mathbf{b}}_i\|} < \sqrt{3/4},$$

where the last inequality follows from the Lovász condition. □

This completes the proof that the number of iterations is $O(N) = \mathrm{poly}(n, |\mathbf{B}|)$. Moreover, each iteration of the algorithm is polynomial time in the bit length of the current basis. However, this does *not* necessarily guarantee that the LLL algorithm is polynomial time overall, since the bit length of the intermediate bases could *increase* with each iteration. (For example, if the bit length doubled in each iteration, then by the end the bit length would be exponential in n.) To it suffices to show that the sizes of all intermediate bases are polynomial in the size of the original basis. This turns out to be the case, due to the size-reduction step. The proof of this fact is somewhat grungy and uninteresting, though, so we won't cover it.

We conclude with some final remarks about the LLL algorithm. The factor $3/4$ in the Lovász condition is just for convenience of analysis. We can use any constant between $1/4$ and 1, which yields a tradeoff between the final approximation factor and the number of iterations, but these will still remain exponential (in n) and polynomial, respectively. By choosing the factor very close to 1, we can obtain an approximation factor of $(2/\sqrt{3})^n$ in polynomial time, but we cannot do any better using LLL. We can get slightly better approximation factors of $2^{O(n(\log\log n)^2)/(\log n)}$ (still in polynomial time) using Schnorr's generalization of LLL, where the analogue of the Lovász condition deals with blocks of $k \geq 2$ consecutive vectors.

2 Coppersmith's Method

One nice application of LLL is a technique of Coppersmith that finds all *small* roots of a polynomial modulo a given number N (even when the factorization of N is unknown). This technique has been a very powerful tool in cryptanalysis, as we will see next time.

Theorem 2.1. *There is an efficient algorithm that, given any monic, degree-d integer polynomial $f(x) \in \mathbb{Z}[x]$ and an integer N, outputs all integers x_0 such that $|x_0| \leq B = N^{1/d}$ and $f(x_0) = 0 \bmod N$.*

We make a few important remarks about the various components of this theorem:

1. When N is prime, i.e., $\mathbb{Z}_N$ is a finite field, there are efficient algorithms that output *all* roots of a given degree-d polynomial $f(x)$ modulo N, of which there are at most d. Similarly, there are efficient algorithm that factor polynomials over the rationals (or integers). Therefore, the fact that the theorem handles a composite modulus N is a distinguishing feature.
2. For composite N, the number of roots of $f(x)$ modulo N can be nearly exponential in the bit length of N, even for quadratic $f(x)$. For example, if N is the product of k distinct primes, then any square modulo N has exactly 2^k distinct square roots. (This follows from the Chinese Remainder Theorem, since there are two square roots modulo each prime divisor of N.) Since k can be as large as $\approx \log N / \log\log N$, the number of roots can be nearly exponential in $\log N$. Therefore, in general no efficient algorithm can output *all* roots of $f(x)$ modulo N; the restriction to *small* roots in the theorem statement circumvents this problem.[1]
3. The size restriction appears necessary for another reason: knowing two square roots $r_1 \neq \pm r_2$ of a square modulo a composite N reveals a nontrivial factor of N, as $\gcd(r_1 - r_2, N)$. So even if the number of roots is small, finding them all is still at least as hard as factoring. However, it is easy to show that a square cannot have more than one "small" square root, of magnitude at most $N^{1/2}$. Therefore, the theorem does appear to yield an efficient factoring algorithm.[2]

To highlight the heart of the method, in the remainder of the section we prove the theorem for a weaker bound of $B \approx N^{2/(d(d+1))}$. (We prove the bound $B \approx N^{1/d}$ next time.) The strategy is to find another nonzero polynomial $h(x) = \sum h_i x^i \in \mathbb{Z}[x]$ such that:

[1] Indeed, the theorem implies that the number of small roots is always polynomially bounded. Surprisingly, this fact did not appear to be known before Coppersmith's result!

[2] However, it can be used to factor when some partial information about a factor is known.

1. every root of $f(x)$ modulo N is also a root of $h(x)$, and
2. the polynomial $h(Bx)$ is "short," i.e., $|h_i B^i| < N/(\deg(h)+1)$ for all i.

For any such $h(x)$, and for any x_0 such that $|x_0| \le B$, we have $|h_i x_0^i| \le |h_i B^i| < N/(\deg(h)+1)$, which implies that $|h(x_0)| < N$. Hence, for every *small* root x_0 (such that $|x_0| \le B$) of $f(x)$ modulo N, we have that $h(x_0) = 0$ *over the integers* (not modulo anything). To find the small roots of $f(x)$ modulo N, we can therefore factor $h(x)$ over the integers, and test whether each of its (small) roots is a root of $f(x)$ modulo N.

We now give an efficient algorithm to find such an $h(x)$. The basic idea is that adding integer multiples of the polynomials $g_i(x) = Nx^i \in \mathbb{Z}[x]$ to $f(x)$ certainly preserves the roots of f modulo N. So we construct a lattice whose basis corresponds to the coefficient vectors of the polynomials $g_i(Bx)$ and $f(Bx)$, find a short nonzero vector in this lattice, and interpret it as the polynomial $h(Bx)$. The lattice basis is

$$\mathbf{B} = \begin{pmatrix} N & & & & a_0 \\ & BN & & & a_1 B \\ & & B^2 N & & a_2 B^2 \\ & & & \ddots & \vdots \\ & & & B^{d-1}N & a_{d-1}B^{d-1} \\ & & & & B^d \end{pmatrix}.$$

Note that the lattice dimension is $d+1$, and that $\det(\mathbf{B}) = B^{d(d+1)/2} \cdot N^d$. By running the LLL algorithm on this basis, we obtain a $2^{d/2}$-approximation $\mathbf{v}$ to a shortest vector in $\mathcal{L}(\mathbf{B})$. By Minkowski's bound,

$$\|\mathbf{v}\| \le 2^{d/2}\sqrt{d+1} \cdot B^{d/2} \cdot N^{d/(d+1)} = c_d \cdot B^{d/2} \cdot N^{1-1/(d+1)},$$

where $c_d = 2^{d/2}\sqrt{d+1}$ depends only on the degree d.

Define $h(Bx)$ to be the polynomial whose coefficients are given by $\mathbf{v}$, i.e., $h(x) = v_0 + (v_1/B)x + \cdots + (v_d/B^d)x^d$. Notice that $h(x) \in \mathbb{Z}[X]$, because B^i divides v_i for each i by construction of the lattice basis, and that every root of $f(x)$ modulo N is also a root of $h(x)$ by construction. Finally, we see that

$$|h_i B^i| = |v_i| \le \|v\| < \frac{N}{d+1},$$

if we take $B < N^{2/d(d+1)}/c'_d$ where $c'_d = (c_d(d+1))^{2/d} = O(1)$ is bounded by a small constant. This concludes the proof.

MATHEMATICS OF OPERATIONS RESEARCH
Vol 8, No. 4, November 1983
Printed in U.S.A.

INTEGER PROGRAMMING WITH A FIXED NUMBER OF VARIABLES*

H. W. LENSTRA, JR.

Universiteit van Amsterdam

It is shown that the integer linear programming problem with a fixed number of variables is polynomially solvable. The proof depends on methods from geometry of numbers.

The *integer linear programming problem* is formulated as follows. Let n and m be positive integers, A an $m \times n$-matrix with integral coefficients, and $b \in \mathbb{Z}^m$. The question is to decide whether there exists a vector $x \in \mathbb{Z}^n$ satisfying the system of m inequalities $Ax \leqslant b$. No algorithm for the solution of this problem is known which has a running time that is bounded by a polynomial function of the *length* of the data. This length may, for our purposes, be defined to be $n \cdot m \cdot \log(a + 2)$, where a denotes the maximum of the absolute values of the coefficients of A and b. Indeed, no such *polynomial algorithm* is likely to exist, since the problem in question is *NP-complete* [3], [12].

In this paper we consider the integer linear programming problem with a fixed value of n. In the case $n = 1$ it is trivial to design a polynomial algorithm for the solution of the problem. For $n = 2$, Hirschberg and Wong [5] and Kannan [6] have given polynomial algorithms in special cases. A complete treatment of the case $n = 2$ was given by Scarf [10]. It was conjectured [5], [10] that for any fixed value of n there exists a polynomial algorithm for the solution of the integer linear programming problem. In the present paper we prove this conjecture by exhibiting such an algorithm. The degree of the polynomial by which the running time of our algorithm can be bounded is an exponential function of n.

Our algorithm is described in §1. Using tools from geometry of numbers [1] we show that the problem can be transformed into an equivalent one having the following additional property: either the existence of a vector $x \in \mathbb{Z}^n$ satisfying $Ax \leqslant b$ is obvious; or it is known that the last coordinate of any such x belongs to an interval whose length is bounded by a constant only depending on n. In the latter case, the problem is reduced to a bounded number of lower dimensional problems.

If in the original problem each coordinate of x is required to be in $\{0, 1\}$, no transformation of the problem is needed to achieve the condition just stated. This suggests that in this case our algorithm is equivalent to complete enumeration. We remark that the $\{0, 1\}$ linear programming problem is NP-complete.

In the general case we need two auxiliary algorithms for the construction of the required transformation. The first of these, which "remodels" the convex set $\{x \in \mathbb{R}^n : Ax \leqslant b\}$, is given in §2. L. Lovász observed that my original algorithm for this could be made polynomial even for varying n, by employing the polynomial solvability of the linear programming problem [8], [4]. I am indebted to Lovász for permission to describe the improved algorithm in §2.

*Received November 13, 1981; revised July 2, 1982.
AMS 1980 subject classification. Primary: 68C25; Secondary: 90C10.
OR/MS Index 1978 subject classification. Primary: 625 Programming/integer/algorithms.
Key words. Integer programming, polynomial algorithm, geometry of numbers.

0364-765X/83/0804/0538$01.25

The second auxiliary algorithm is a reduction process for n-dimensional lattices. Such an algorithm, also due to Lovász, appeared in [9, §1], and a brief sketch is given in §3 of the present paper. This algorithm is polynomial even for varying n. It supersedes the much inferior algorithm that was described in an earlier version of this paper.

In §4 we prove, following a suggestion of P van Emde Boas, that the integer linear programming problem with a fixed value of m is also polynomially solvable This is an immediate consequence of our main result.

§5 is devoted to the *mixed integer linear programming problem*. Combining our methods with Khachiyan's results [8], [4] we show that this problem is polynomially solvable for any fixed value of the number of integer variables This generalizes both our main result and Khachiyan's theorem

The algorithms presented in this paper were designed for theoretical purposes only, and there are several modifications that might improve their practical performance. It is to be expected that the practical value of our algorithms is restricted to small values of n.

It is a pleasure to acknowledge my indebtedness to P. van Emde Boas, not only for permission to include §4, but also for suggesting the problem solved in this paper and for several inspiring and stimulating discussions.

1. Description of the algorithm. Let K denote the closed convex set

$$K = \{x \in \mathbb{R}^n : Ax \leqslant b\}$$

The question to be decided is whether $K \cap \mathbb{Z}^n = \emptyset$ In the description of the algorithm that follows, we make the following two simplifying assumptions about K:

(1) K is *bounded*;

(2) K has *positive volume*.

The first assumption is justified by the following result, which is obtained by combining a theorem of Von zur Gathen and Sieveking [12] with Hadamard's determinant inequality (cf. (6) below): the set $K \cap \mathbb{Z}^n$ is nonempty if and only if $K \cap \mathbb{Z}^n$ contains a vector whose coefficients are bounded by $(n+1)n^{n/2}a^n$ in absolute value, where a is as in the introduction. Adding these inequalities to the system makes K bounded.

For the justification of condition (2) we refer to §2. Under the assumptions (1) and (2), §2 describes how to construct a nonsingular endomorphism τ of the vector space $\mathbb{R}^n$, such that τK has a "spherical" appearance. More precisely, let $|\ |$ denote the Euclidean length in $\mathbb{R}^n$, and put

$$B(p,z) = \{x \in \mathbb{R}^n : |x-p| \leqslant z\} \qquad \text{for} \quad p \in \mathbb{R}^n, \quad z \in \mathbb{R}_{>0},$$

the closed ball with center p and radius z. With this notation, the τ constructed will satisfy

$$B(p,r) \subset \tau K \subset B(p,R) \tag{3}$$

for some $p \in \tau K$, with r and R satisfying

$$\frac{R}{r} \leqslant c_1, \tag{4}$$

where c_1 is a constant only depending on n.

Let such a τ be fixed, and put $L = \tau\mathbb{Z}^n$. This is a *lattice* in $\mathbb{R}^n$, i e , there exists a basis $b_1, b_2, \ldots, b_n$ of $\mathbb{R}^n$ such that

$$L = \sum_{i=1}^{n} \mathbb{Z}b_i = \left\{ \sum_{i=1}^{n} m_i b_i : m_i \in \mathbb{Z}\ (1 \leqslant i \leqslant n) \right\} \tag{5}$$

We can take, for example, $b_i = \tau(e_i)$, with e_i denoting the ith standard basis vector of $\mathbb{R}^n$ We call $b_1, b_2, \ldots, b_n$ a *basis* for L if (5) holds. If $b'_1, b'_2, \ldots, b'_n$ is another basis for L, then $b'_i = \sum_{j=1}^{n} m_{ij} b_j$ for some $n \times n$-matrix $M = (m_{ij})_{1 \leqslant i,j \leqslant n}$ with integral coefficients and $\det(M) = \pm 1$. It follows that the positive real number $|\det(b_1, b_2, \ldots, b_n)|$ (the b_i being written as column vectors) only depends on L, and not on the choice of the basis; it is called the *determinant* of L, notation: $d(L)$. We can interpret $d(L)$ as the volume of the parallelepiped $\sum_{i=1}^{n} [0,1) \cdot b_i$, where $[0,1) = \{z \in \mathbb{R} \cdot 0 \leqslant z < 1\}$. This interpretation leads to the *inequality of Hadamard*

$$d(L) \leqslant \prod_{i=1}^{n} |b_i|. \tag{6}$$

The equality sign holds if and only if the basis $b_1, b_2, \ldots, b_n$ is orthogonal. It is a classical theorem that L has a basis $b_1, b_2, \ldots, b_n$ that is nearly orthogonal in the sense that the following inequality holds:

$$\prod_{i=1}^{n} |b_i| \leqslant c_2 \cdot d(L) \tag{7}$$

where c_2 is a constant only depending on n, cf. [1, Chapter VIII], [11]. In §3 we shall indicate a *reduction process*, i.e., an algorithm that changes a given basis for L into one satisfying (7).

LEMMA. *Let $b_1, b_2, \ldots, b_n$ be any basis for L. Then*

$$\forall x \in \mathbb{R}^n : \exists y \in L : |x - y|^2 \leqslant \tfrac{1}{4}(|b_1|^2 + \cdots + |b_n|^2). \tag{8}$$

PROOF We use induction on n, the case $n = 1$ (or $n = 0$) being obvious. Let $L' = \sum_{i=1}^{n-1} \mathbb{Z} b_i$, this is a lattice in the $(n-1)$-dimensional hyperplane $H = \sum_{i=1}^{n-1} \mathbb{R} b_i$. Denote by h the distance of b_n to H. Clearly we have

$$h \leqslant |b_n|. \tag{9}$$

Now to prove (8), let $x \in \mathbb{R}^n$. We can find $m \in \mathbb{Z}$ such that the distance of $x - mb_n$ to H is $\leqslant \frac{1}{2} h$. Write $x - mb_n = x_1 + x_2$, with $x_1 \in H$ and x_2 perpendicular to H. Then $|x_2| \leqslant \frac{1}{2} h \leqslant \frac{1}{2} |b_n|$. By the induction hypothesis there exists $y_1 \in L'$ such that $|x_1 - y_1|^2 \leqslant \frac{1}{4}(|b_1|^2 + \cdots + |b_{n-1}|^2)$. Since x_2 is orthogonal to y_1 the element $y = y_1 + mb_n$ of L now satisfies $|x - y|^2 = |x_1 - y_1|^2 + |x_2|^2 \leqslant \frac{1}{4}(|b_1|^2 + \cdots + |b_{n-1}|^2 + |b_n|^2)$. This proves the lemma.

Notice that the proof gives an effective construction of the element $y \in L$ that is asserted to exist.

If we number the b_i such that $|b_n| = \max\{|b_i| : 1 \leqslant i \leqslant n\}$, then (8) implies

$$\forall x \in \mathbb{R}^n : \exists y \in L : |x - y| \leqslant \tfrac{1}{2}\sqrt{n}\,|b_n|. \tag{10}$$

Now assume that $b_1, b_2, \ldots, b_n$ is a *reduced* basis for L in the sense that (7) holds, and let L' and h have the same meaning as in the proof of the lemma. It is easily seen that

$$d(L) = h \cdot d(L'). \tag{11}$$

From (7), (11) and (6), applied to L', we get

$$\prod_{i=1}^{n} |b_i| \leqslant c_2 \cdot d(L) = c_2 \cdot h \cdot d(L') \leqslant c_2 \cdot h \cdot \prod_{i=1}^{n-1} |b_i|$$

and therefore, with (9):

$$c_2^{-1}\ |b_n| \leqslant h \leqslant |b_n|. \tag{12}$$

After these preparations we describe the procedure by which we decide whether $K \cap \mathbb{Z}^n = \emptyset$ or, equivalently, $\tau K \cap L = \emptyset$. We assume that $b_1, b_2, \ldots, b_n$ is a basis for L for which (7) holds, numbered such that $|b_n| = \max\{|b_i| : 1 \leqslant i \leqslant n\}$.

Applying (10) with $x = p$ we find a vector $y \in L$ with $|p - y| \leqslant \frac{1}{2}\sqrt{n}\,|b_n|$. If $y \in \tau K$ then $\tau K \cap L \neq \emptyset$, and we are done. Suppose therefore that $y \notin \tau K$. Then $y \notin B(p,r)$, by (3), so $|p - y| > r$, and this implies that $r < \frac{1}{2}\sqrt{n}\,|b_n|$. Let now H, L', h have the same meaning as in the proof of the lemma. We have

$$L = L' + \mathbb{Z}b_n \subset H + \mathbb{Z}b_n = \bigcup_{k \in \mathbb{Z}} (H + kb_n).$$

Hence L is contained in the union of countably many parallel hyperplanes, which have successive distances h from each other. We are only interested in those hyperplanes that have a nonempty intersection with τK, these have, by (3), also a nonempty intersection with $B(p, R)$. Suppose that precisely t of the hyperplanes $H + kb_n$ intersect $B(p, R)$. Then we have clearly $t - 1 \leqslant 2R/h$. By (4) and (12) we have

$$2R \leqslant 2rc_1 < c_1\sqrt{n}\,|b_n|, \qquad h \geqslant c_2^{-1}|b_n|$$

so $t - 1 < c_1c_2\sqrt{n}$. Hence the number of values for k that have to be considered is bounded by a constant only depending on n. Which values of k need be considered can easily be deduced from a representation of p as a linear combination of b_1, $b_2, \ldots, b_n$.

If we fix the value of k then we restrict attention to those $x = \sum_{i=1}^{n} y_i b_i$ for which $y_n = k$; and this leads to an integer programming problem with $n - 1$ variables $y_1, y_2, \ldots, y_{n-1}$. It is straightforward to show that the length of the data of this new problem is bounded by a polynomial function of the length of the original data, if the directions of §2 have been followed for the construction of τ.

Each of the lower dimensional problems is treated recursively. The case of dimension $n = 1$ (or even $n = 0$) may serve as a basis for the recursion. This finishes our description of the algorithm.

We observe that in the case that $K \cap \mathbb{Z}^n$ is nonempty, our algorithm actually produces an element $x \in K \cap \mathbb{Z}^n$.

2. The convex set K. Let $K = \{x \in \mathbb{R}^n : Ax \leqslant b\}$, and assume that K is bounded. In this section we describe an algorithm that can be used to verify that K satisfies condition (2) of §1; to reduce the number of variables if that condition is found not to be satisfied; and to find the map τ used in §1. The algorithm is better than what is strictly needed in §1, in the sense that it is polynomial even for varying n. I am indebted to L. Lovász for pointing out to me how this can be achieved.

In the first stage of the algorithm one attempts to construct vertices $v_0, v_1, \ldots, v_n$ of K whose convex hull is an n-simplex of *positive* volume. By maximizing an arbitrary linear function on K, employing Khachiyan's algorithm [8], [4], one finds a vertex v_0 of K, unless K is empty. Suppose, inductively, that vertices $v_0, v_1, \ldots, v_d$ of K have been found for which $v_1 - v_0, \ldots, v_d - v_0$ are linearly independent, with $d < n$. Then we can construct $n - d$ linearly independent linear functions $f_1, \ldots, f_{n-d}$ on $\mathbb{R}^n$ such that the d-dimensional subspace $V = \sum_{j=1}^{d} \mathbb{R}(v_j - v_0)$ is given by

$$V = \{x \in \mathbb{R}^n : f_1(x) = \cdots = f_{n-d}(x) = 0\}.$$

Again employing Khachiyan's algorithm, we maximize each of the linear functions $f_1, -f_1, f_2, -f_2, \ldots, f_{n-d}, -f_{n-d}$ on K, until a vertex v_{d+1} of K is found for which $f_j(v_{d+1}) \neq f_j(v_0)$ for some $j \in \{1, 2, \ldots, n-d\}$. If this occurs, then $v_1 - v_0, \ldots, v_d - v_0, v_{d+1} - v_0$ are linearly independent, and the inductive step of the construction is completed. If, on the other hand, no such v_{d+1} is found after each of the $2(n-d)$ functions $f_1, -f_1, \ldots, f_{n-d}, -f_{n-d}$ has been maximized, then we must have $f_j(x) = f_j(v_0)$ for all $x \in K$ and all $j = 1, 2, \ldots, n-d$, and therefore $K \subset v_0 + V$. In this case we reduce the problem to an integer programming problem with only d variables, as follows.

Choose, for $j = 1, 2, \ldots, d$, a nonzero scalar multiple w_j of $v_j - v_0$ such that $w_j \in \mathbb{Z}^n$, and denote by W the $(n \times d)$-matrix whose columns are the w_j. Notice that W has rank d. Employing the *Hermite normal form* algorithm of Kannan and Bachem [7] we can find, in polynomial time, an integral $n \times n$-matrix U with $\det(U) = \pm 1$ such that

$$UW = (k_{ij})_{1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant d}$$

with

$$\begin{cases} k_{ij} = 0 & \text{if } \ i > j, \\ k_{ii} \neq 0 & \text{for } \ 1 \leqslant i \leqslant d. \end{cases} \tag{13}$$

Denote by $u_1, u_2, \ldots, u_n$ the columns of the integral matrix U^{-1}. These form a basis of $\mathbb{R}^n$, and also of the lattice $\mathbb{Z}^n$: $\mathbb{Z}^n = \sum_{j=1}^n \mathbb{Z}u_j$. The subspace V of $\mathbb{R}^n$ is generated by the columns of $W = U^{-1} \cdot (k_{ij})$, so (13) implies that

$$V = \sum_{j=1}^{d} \mathbb{R}u_j. \tag{14}$$

Define $r_1, r_2, \ldots, r_n \in \mathbb{R}$ by $v_0 = \sum_{j=1}^n r_j u_j$; so $(r_j)_{j=1}^n = Uv_0$.

Now suppose that $x \in K \cap \mathbb{Z}^n$. Then $x = \sum_{j=1}^n y_j u_j$ with $y_j \in \mathbb{Z}$, and $x \in K$ implies that $x - v_0 \in V$. By (14) this means that $y_j = r_j$ for $d < j \leqslant n$. So if at least one of $r_{d+1}, \ldots, r_n$ is not an integer, then $K \cap \mathbb{Z}^n = \emptyset$. Suppose, therefore, that $r_{d+1}, \ldots, r_n$ are all integral. Substituting $x = \sum_{j=1}^d y_j u_j + \sum_{j=d+1}^n r_j u_j$ in our original system $Ax \leqslant b$ we then see that the problem is equivalent to an integer programming problem with d variables $y_1, y_2, \ldots, y_d$, as required. The vertices $v_0, v_1, \ldots, v_d$ of K give rise to $d+1$ vertices $v_0', v_1', \ldots, v_d'$ of the convex set in $\mathbb{R}^d$ belonging to the new problem, and $v_0', v_1', \ldots, v_d'$ span a d-dimensional simplex of positive volume. This means that for the new, d-dimensional problem the first stage of the algorithm that we are describing can be bypassed.

To conclude the first stage of the algorithm, we may now suppose that for each $d = 0, 1, \ldots, n-1$ the construction of v_{d+1} is successful. Then after n steps we have $n+1$ vertices $v_0, v_1, \ldots, v_n$ of K for which $v_1 - v_0, \ldots, v_n - v_0$ are linearly independent. The n-simplex spanned by $v_0, v_1, \ldots, v_n$ is contained in K, and its volume equals $|\det M|/n!$ where M is the matrix with column vectors $v_1 - v_0, \ldots, v_n - v_0$. This is positive, so condition (2) of §1 is satisfied.

In the second stage of the algorithm we construct the coordinate transformation τ needed in §1. To this end we first try to find a simplex of "large" volume in K. This is done by an iterative application of the following procedure, starting from the simplex spanned by $v_0, v_1, \ldots, v_n$. The volume of that simplex is denoted by $\mathrm{vol}(v_0, v_1, \ldots, v_n)$.

Construct $n+1$ linear functions $g_0, g_1, \ldots, g_n : \mathbb{R}^n \to \mathbb{R}$ such that

$$\begin{aligned} &g_i \text{ is constant on } \{v_j : 0 \leqslant j \leqslant n, j \neq i\}, \\ &g_i(v_i) \neq g_i(v_j) \qquad \text{for } \ 0 \leqslant j \leqslant n, j \neq i, \end{aligned} \tag{15}$$

for $i = 0, 1, \ldots, n$. Maximizing the functions $g_0, -g_0, g_1, -g_1, \ldots, g_n, -g_n$ on K by Khachiyan's algorithm we can decide whether there exist $i \in \{0, 1, \ldots, n\}$ and a vertex x of K such that

$$|g_i(x - v_j)| > \tfrac{3}{2}|g_i(v_i - v_j)|$$

for $j \neq i$ (the choice of j is immaterial, by (15)).

Suppose that such a pair i, x is found. Then we replace v_i by x. This replacement enlarges $\mathrm{vol}(v_0, v_1, \ldots, v_n)$ by a factor $|g_i(x - v_j)|/|g_i(v_i - v_j)|$ (for $j \neq i$), which is more than $3/2$. We now return to the beginning of the procedure ("Construct $n + 1$ linear functions . . . ").

In every iteration step $\mathrm{vol}(v_0, v_1, \ldots, v_n)$ increases by a factor $> 3/2$. On the other hand, this volume is bounded by the volume of K. Hence after a polynomially bounded number of iterations we reach a situation in which the above procedure discovers that

$$|g_i(x - v_j)| \leqslant \tfrac{3}{2}|g_i(v_i - v_j)| \tag{16}$$

for all $x \in K$ and all $i, j \in \{0, 1, \ldots, n\}$ with $i \neq j$. In that case we let τ be a nonsingular endomorphism of $\mathbb{R}^n$ with the property that $\tau(v_0), \tau(v_1), \ldots, \tau(v_n)$ span a *regular* n-simplex. With $p = (n+1)^{-1}\sum_{j=0}^{n}\tau(v_j)$ we now claim that $B(p, r) \subset \tau K \subset B(p, R)$ for certain positive real numbers r, R satisfying $R/r \leqslant 2n^{3/2}$, i.e., that conditions (3) and (4) of §1 are satisfied, with $c_1 = 2n^{3/2}$. This finishes the description of our algorithm.

To prove our claim, we write $z_j = \tau(v_j)$, for $0 \leqslant j \leqslant n$; we write S for the regular n-simplex spanned by $z_0, z_1, \ldots, z_n$, and we define, for $c \geqslant 1$:

$$T_c = \{x \in \mathbb{R}^n : \mathrm{vol}(z_0, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n)$$
$$\leqslant c \cdot \mathrm{vol}(z_0, \ldots, z_n) \text{ for all } i \in \{0, 1, \ldots, n\}\}.$$

Condition (16) (for all $x \in K$ and all $i \neq j$) means precisely that $\tau K \subset T_{3/2}$. Further, it is clear that $S \subset \tau K$. Our claim now follows from the following lemma.

LEMMA. *Let $c \geqslant 1$. With the above notation we have $B(p, r) \subset S \subset T_c \subset B(p, R)$ for two positive real numbers r, R satisfying*

$$\left(\frac{R}{r}\right)^2 = \begin{cases} c^2n^3 + (c^2+1)n^2 & \textit{if } n \textit{ is even,} \\ c^2n^3 + (2c^2 - 2c + 1)n^2 + (c^2 - 2c)n & \textit{if } n \textit{ is odd.} \end{cases}$$

PROOF. Using a similarity transformation we can identify $\mathbb{R}^n$ with the hyperplane $\{(r_j)_{j=0}^n \in \mathbb{R}^{n+1} : \sum_{j=0}^n r_j = 1\}$ in $\mathbb{R}^{n+1}$ such that $z_0, z_1, \ldots, z_n$ is the standard basis of $\mathbb{R}^{n+1}$. Then we have

$$p = \frac{1}{n+1}\sum_{j=0}^{n} z_j = \left(\frac{1}{n+1}, \frac{1}{n+1}, \ldots, \frac{1}{n+1}\right),$$

and

$$T_c = \left\{(r_j)_{j=0}^n \in \mathbb{R}^{n+1} : |r_j| \leqslant c \text{ for } 0 \leqslant j \leqslant n, \text{ and } \sum_{j=0}^{n} r_j = 1\right\}.$$

By a straightforward analysis one proves that T_c is the convex hull of the set of points obtained by permuting the coordinates of the point

$$z_0 - c\sum_{j=1}^{m} z_j + c\sum_{j=m+1}^{n} z_j \qquad \text{if} \quad n = 2m,$$

$$(1-c)z_0 - c\sum_{j=1}^{m} z_j + c\sum_{j=m+1}^{n} z_j \qquad \text{if} \quad n = 2m+1.$$

It follows that $T_c \subset B(p,R)$, where R is the distance of p to the above point:

$$R^2 = \begin{cases} nc^2 + \dfrac{n}{n+1} & \text{if } n \text{ is even,} \\ (n+1)c^2 - 2c + \dfrac{n}{n+1} & \text{if } n \text{ is odd.} \end{cases}$$

Further, $B(p,r) \subset S$, where r is the distance of p to $(0, 1/n, 1/n, \ldots, 1/n)$:

$$r^2 = \frac{1}{n(n+1)}.$$

This proves the lemma.

REMARKS. (a) To the construction of τ in the above algorithm one might raise the objection that τ need not be given by a matrix with *rational* coefficients. Indeed, for $n = 2,4,5,6,10,\ldots$ there exists no regular n-simplex all of whose vertices have rational coordinates. This objection can be answered in several ways. One might replace the regular simplex by a rational approximation of it, or indeed by any fixed n-simplex with rational vertices and positive volume, at the cost of getting a larger value for c_1. Alternatively, one might embed $\mathbb{R}^n$ in $\mathbb{R}^{n+1}$, as was done in the proof of the lemma. Finally, it can be argued that it is not necessary that the matrix M_τ defining τ be rational, but only the symmetric matrix $M_\tau{}^{\mathsf{T}} M_\tau$ defining the quadratic form $(\tau x, \tau x)$; and this can easily be achieved in the above construction of τ.

(b) The proof that the algorithm described in this section is polynomial, even for varying n, is entirely straightforward. We indicate the main points. The construction of $f_1, \ldots, f_{n-d}$ in the first stage, and of $g_0, g_1, \ldots, g_n$ in the second stage, can be done by Gaussian elimination, which is well known to be a polynomial algorithm, cf. [2, §7]. It follows that Khachiyan's algorithm is only applied to problems whose lengths are bounded by a polynomial function of the length of the original data. The same applies to the d-dimensional integer programming problem constructed in the first stage. Further details are left to the reader.

(c) We discuss to which extent the value $2n^{3/2}$ for c_1 in (4) is best possible. Replacing the coefficient 3/2 in (16) by other constants $c > 1$ we find, using the lemma, that for any fixed $\epsilon > 0$ we can take

$$c_1 = \begin{cases} (1+\epsilon)(n^3 + 2n^2)^{1/2} & \text{if } n \text{ is even,} \\ (1+\epsilon)(n^3 + n^2 - n)^{1/2} & \text{if } n \text{ is odd.} \end{cases}$$

If one is satisfied with an algorithm that is only polynomial for fixed n one can also take $\epsilon = 0$ in this formula. To achieve this, one uses a list of all vertices of K to find the simplex of maximal volume inside K, and transforms this simplex into a regular one. The following result shows that there is still room for improvement: if $K \subset \mathbb{R}^n$ is any closed convex set satisfying (1) and (2) then there exists a nonsingular endomorphism τ of $\mathbb{R}^n$ such that (3) and (4) hold with $c_1 = n$. To prove this, one chooses an *ellipsoid* E inside K with maximal volume, and one chooses τ such that τE is a sphere. The case that K is a simplex shows that the value $c_1 = n$ is best possible. For fixed n and $\epsilon > 0$ there is a polynomial algorithm that achieves $c_1 = (1+\epsilon)n$. I do not know how well the best possible value $c_1 = n$ can be approximated by an algorithm that is polynomial for varying n.

(d) The algorithm described in this section applies equally well to any class $\mathscr{K}$ of compact convex bodies in $\mathbb{R}^n$ for which there exists a polynomial algorithm that maximizes linear functions on members K of $\mathscr{K}$. This remark will play an important role in §5. In particular, we can take for $\mathscr{K}$ a "solvable" class of convex bodies, in the terminology of [4, §§1 and 3]. The same remark can be made for the algorithm presented in §1.

3. The reduction process. Let n be a positive integer, and let $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$ be n linearly independent vectors. Put $L = \sum_{i=1}^n \mathbb{Z} b_i$; this is a lattice in $\mathbb{R}^n$. In this section we indicate an algorithm that transforms the basis $b_1, b_2, \ldots, b_n$ for L into one satisfying (7) with $c_2 = 2^{n(n-1)/4}$ The algorithm is taken from [9, §1], to which we refer for a more detailed description.

We recall the Gram-Schmidt orthogonalization process. The vectors b_i^* $(1 \leqslant i \leqslant n)$ and the real numbers μ_{ij} $(1 \leqslant j < i \leqslant n)$ are inductively defined by

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \qquad \mu_{ij} = (b_i, b_j^*)/(b_j^*, b_j^*),$$

where (,) denotes the ordinary inner product on $\mathbb{R}^n$. Notice that b_i^* is the projection of b_i on the orthogonal complement of $\sum_{j=1}^{i-1} \mathbb{R} b_j$, and that $\sum_{j=1}^{i-1} \mathbb{R} b_j = \sum_{j=1}^{i-1} \mathbb{R} b_j^*$, for $1 \leqslant i \leqslant n$. It follows that $b_1^*, b_2^*, \ldots, b_n^*$ is an orthogonal basis of $\mathbb{R}^n$. The following result is taken from [9].

PROPOSITION. *Suppose that*

$$|\mu_{ij}| \leqslant \tfrac{1}{2} \tag{17}$$

for $1 \leqslant j < i \leqslant n$, *and*

$$|b_i^* + \mu_{i\,i-1} b_{i-1}^*|^2 \geqslant \tfrac{3}{4} |b_{i-1}^*|^2 \tag{18}$$

for $1 < i \leqslant n$. *Then*

$$\prod_{i=1}^n |b_i| \leqslant 2^{n(n-1)/4} d(L),$$

i.e., (7) *holds with* $c_2 = 2^{n(n-1)/4}$.

PROOF. See [9, Proposition 1.6].

To explain condition (18) we remark that the vectors $b_i^* + \mu_{i\,i-1} b_{i-1}^*$ and b_{i-1}^* are the projections of b_i and b_{i-1} on the orthogonal complement of $\sum_{j=1}^{i-2} \mathbb{R} b_j$. Hence if (18) does not hold for some i, then it does hold for the basis obtained from $b_1, b_2, \ldots, b_n$ by interchanging b_{i-1} and b_i.

To change a given basis $b_1, b_2, \ldots, b_n$ for L into one satisfying (7) we may now iteratively apply the following transformations.

First transformation: select i, $1 < i \leqslant n$, such that (18) does not hold, and interchange b_{i-1} and b_i;

Second transformation: select i, j, $1 \leqslant j < i \leqslant n$, such that (17) does not hold, and replace b_i by $b_i - r b_j$, where r is the integer nearest to μ_{ij}.

It can be shown that, independently of the order in which these transformations are applied and independently of the choices of i, and of i and j, that are made, this leads after a finite number of steps to a basis $b_1, b_2, \ldots, b_n$ satisfying (17) and (18). Then (7) is satisfied as well, by the proposition. This finishes our sketch of the algorithm.

A particularly efficient strategy for choosing which transformation to apply, and for which i, or i and j, is described in [9, (1.15)]. If we assume the b_i to have *integer* coordinates then the resulting algorithm is polynomial, even for varying n, by [9, Proposition 1.26]. It follows that the same result is true if we allow the coordinates of the b_i to be *rational*.

REMARKS. (a) The algorithm sketched above can be used to find the shortest nonzero vector in L, in the following way. Suppose that $b_1, b_2, \ldots, b_n$ is a basis for L satisfying (7), and let $x \in L$. Then we can write $x = \sum_{i=1}^{n} m_i b_i$ with $m_i \in \mathbb{Z}$, and from Cramer's rule it is easy to derive that $|m_i| \leqslant c_2 \cdot |x|/|b_i|$, for $1 \leqslant i \leqslant n$. If x is the shortest nonzero vector in L then $|x| \leqslant |b_i|$ for all i, so $|m_i| \leqslant c_2$. So by searching the set $\{\sum_{i=1}^{n} m_i b_i : m_i \in \mathbb{Z},\ |m_i| \leqslant c_2 \text{ for } 1 \leqslant i \leqslant n\}$ we can find the shortest nonzero vector in L in polynomial time, for fixed n. For variable n this problem is likely to be NP-hard.

(b) We discuss to which extent our value for c_2 is best possible. The above algorithm yields $c_2 = 2^{n(n-1)/4}$. We indicate an algorithm that leads to a much better value for c_2; but the algorithm is only polynomial for fixed n.

In (a) we showed how to find the shortest nonzero vector in L by a search procedure. By an analogous but somewhat more complicated search procedure we can determine the *successive minima* $|b_1'|, |b_2'|, \ldots, |b_n'|$ of L (see [1, Chapter VIII] for the definition). Here $b_1', b_2', \ldots, b_n' \in L$ are linearly independent, and by [1, Chapter VIII, Theorem I, p. 205 and Chapter IV, Theorem VII, p. 120] they satisfy

$$\prod_{i=1}^{n} |b_i'| \leqslant \gamma_n^{n/2} \cdot d(L)$$

where γ_n denotes Hermite's constant [1, §IX.7, p. 247], for which it is known that

$$\frac{1}{2\pi e} + o(1) \leqslant \gamma_n/n \leqslant \frac{1}{\pi e} + o(1) \qquad \text{for} \quad n \to \infty.$$

Using a slight improvement of [1, Chapter V, Lemma 8, p. 135] we can change $b_1', b_2', \ldots, b_n'$ into a *basis* $b_1'', b_2'', \ldots, b_n''$ for L satisfying

$$|b_i''| \leqslant \max\left\{1, \tfrac{1}{2}\sqrt{i}\right\} \cdot |b_i'| \qquad (1 \leqslant i \leqslant n)$$

so

$$\prod_{i=1}^{n} |b_i''| \leqslant 2^{-n+2} \cdot \left(\tfrac{2}{3} n!\right)^{1/2} \cdot \gamma_n^{n/2} \cdot d(L) \qquad (\text{for } n \geqslant 3).$$

We conclude that, for fixed n, the basis $b_1, b_2, \ldots, b_n$ produced by the algorithm indicated in this section can be used to find, in polynomial time, a new basis satisfying (7), but now with $c_2 = (c \cdot n)^n$. Here c denotes some absolute positive constant.

On the other hand, the definition of γ_n implies that there exists an n-dimensional lattice L such that $|x| \geqslant \gamma_n^{1/2} \cdot d(L)^{1/n}$ for all $x \in L$, $x \neq 0$, cf. [1, Chapter I, Lemma 4, p. 21]. Any basis $b_1, b_2, \ldots, b_n$ for such a lattice clearly satisfies

$$\prod_{i=1}^{n} |b_i| \geqslant \gamma_n^{n/2} \cdot d(L).$$

Therefore the best possible value for c_2 satisfies $c_2 > (c' \cdot n)^{n/2}$ for some absolute positive constant c'.

4. A fixed number of constraints.

In this section we show that the integer linear programming problem with a fixed value of m is polynomially solvable. It was noted by P. van Emde Boas that this is an immediate consequence of our main result.

Let n, m, A, b be as in the introduction. We have to decide whether there exists $x \in \mathbb{Z}^n$ for which $Ax \leqslant b$. Applying the algorithms of Kannan and Bachem [7] we can find an $(n \times n)$-matrix U with integral coefficients and determinant ± 1 such that the matrix $AU = (a_{ij}')_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n}$ satisfies

$$a_{ij}' = 0 \qquad \text{for} \quad j > i. \tag{19}$$

Putting $y = U^{-1}x$ we see that the existence of $x \in \mathbb{Z}^n$ with $Ax \leqslant b$ is equivalent to the existence of $y \in \mathbb{Z}^n$ with $(AU)y \leqslant b$. If $n > m$, then the coordinates $y_{m+1}, \ldots, y_n$ of y do not occur in these inequalities, since (19) implies that $a'_{ij} = 0$ for $j > m$. We conclude that the original problem can be reduced to a problem with only $\min\{n,m\}$ variables. The latter problem is, for fixed m, polynomially solvable, by the main result of this paper.

5. Mixed integer linear programming. The *mixed integer linear programming problem* is formulated as follows. Let k and m be positive integers, and n an integer satisfying $0 \leqslant n \leqslant k$. Let further A be an $m \times k$-matrix with integral coefficients, and $b \in \mathbb{Z}^m$. The question is to decide whether there exists a vector $x = (x_1, x_2, \ldots, x_k)^{\mathrm{T}}$ with

$$x_i \in \mathbb{Z} \quad \text{for} \quad 1 \leqslant i \leqslant n,$$

$$x_i \in \mathbb{R} \quad \text{for} \quad n+1 \leqslant i \leqslant k$$

satisfying the system of m inequalities $Ax \leqslant b$.

In this section we indicate an algorithm for the solution of this problem that is polynomial for any fixed value of n, the number of integer variables. This generalizes both the result of §1 ($n = k$) and the result of Khachiyan [8], [4] ($n = 0$).

Let

$$K' = \{x \in \mathbb{R}^k : Ax \leqslant b\},$$

$$K = \{(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : \text{there exist } x_{n+1}, \ldots, x_k \in \mathbb{R} \text{ such that } (x_1, x_2, \ldots, x_k) \in K'\}.$$

The question is whether $K \cap \mathbb{Z}^n = \emptyset$.

Making use of the arguments of Von zur Gathen and Sieveking [12] we may again assume that K', and hence K, is bounded. Next we apply the algorithm of §2 to the compact convex set $K \subset \mathbb{R}^n$. To see that this can be done it suffices to show that we can maximize linear functions on K, see §2, Remark (d). But maximizing linear functions on K is equivalent to maximizing, on K', linear functions that depend only on the first n coordinates $x_1, x_2, \ldots, x_n$; and this can be done by Khachiyan's algorithm.

The rest of the algorithm proceeds as before. At a certain point in the algorithm we have to decide whether a given vector $y \in \mathbb{R}^n$ belongs to τK. This can be done by solving a linear programming problem with $k - n$ variables. This finishes the description of the algorithm.

As in §4 it can be proved that the mixed integer linear programming problem is also polynomially solvable if the number of inequalities that involve one or more integer variables is fixed; or, more generally, if the rank of the matrix formed by the first n columns of A is bounded.

References

[1] Cassels, J. W. S. (1959). *An Introduction to the Geometry of Numbers*. Springer, Berlin. Second printing, 1971.

[2] Edmonds, J. (1967). Systems of Distinct Representatives and Linear Algebra. *J. Res. Nat. Bur. Standards Sect. B* **71B** 241–245.

[3] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman & Co., San Francisco.

[4] Grotschel, M., Lovász, L. and Schrijver, A. (1981). The Ellipsoid Method and Its Consequences in Combinatorial Optimization *Combinatorica* **1** 169–197.

[5] Hirschberg, D. S. and Wong, C. K. (1976). A Polynomial-time Algorithm for the Knapsack Problem with Two Variables. *J. Assoc. Comput. Mach.* **23** 147–154

[6] Kannan, R. (1980). A Polynomial Algorithm for the Two-variable Integer Programming Problem *J. Assoc. Comput Mach* **27** 118–122

[7] ——— and Bachem, A. (1979) Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix. *SIAM J. Comput.* **8** 499–507

[8] Khachiyan, L. G (1979). A Polynomial Algorithm in Linear Programming *Dokl. Akad Nauk SSSR* **244** 1093–1096 (English translation: *Soviet Math Dokl.* **20** (1979), 191–194)

[9] Lenstra, A. K., Lenstra, H. W., Jr. and Lovász, L (1982) Factoring Polynomials with Rational Coefficients. *Math. Ann.* **261** 515–534.

[10] Scarf, H. E. (1981). Production Sets with Indivisibilities—Part I Generalities *Econometrica* **49** 1–32 Part II The Case of Two Activities, ibid, 395–423.

[11] Van der Waerden, B. L. (1956) Die Reduktionstheorie von positiven quadratischen Formen *Acta Math* **96** 265–309.

[12] Von zur Gathen, J. and Sieveking, M. (1978). A Bound on Solutions of Linear Integer Equalities and Inequalities *Proc Amer. Math. Soc.* **72** 155–158

MATHEMATISCH INSTITUUT, UNIVERSITEIT VAN AMSTERDAM, ROETERSSTRAAT 15, 1018 WB AMSTERDAM, THE NETHERLANDS

Improved algorithms for integer programming and related lattice problems

by

Ravi Kannan
M.I.T.
Cambridge, MA. 02139

ABSTRACT:

The integer programming problem is: Given $m \times n$ and $m \times 1$ matrices A and b respectively of integers, find whether there exists an all integer $n \times 1$ vector x satisfying the m inequalities $Ax \le b$. In settling an important open problem, Lenstra (1981) showed in an elegant way that when n, the number of dimensions is fixed, there is a polynomial-time algorithm to solve this problem. His algorithm achieves a running-time of $O(c^{n^3} \cdot p(\text{length of data}))$ where p is some polynomial and c a constant independent of n. Since such an algorithm has several important applications - cryptography (Shamir (1982)), diophantine approximations (Lagarias (1982)), coding theory (Conway and Sloane (1982), etc. it is important to improve the running time. We present an algorithm here that has a running time of $O(n^{9n} L \log L)$ where L is the length of the input. Whereas Lenstra's algorithm in the worst case reduces an n-dimensional problem to c^{n^2}-(n-1) dimensional problems, our algorithm effectively reduces an n-dimensional problem to at most polynomially many (n-1) dimensional problems, thus achieving our time bound. The algorithm we propose, first finds a "more orthogonal" basis for a lattice (see the next section for the definition of a lattice) than those of Lenstra (1981) and Lenstra, Lenstra and Lovasz (1982), but in time $O(n^{dn}$ poly (length of input)). It then uses an enumeration technique to solve integer programming and related problems. The proof that only $O(n^{d'n})$ possibilities need to be tried for this enumeration is based on Minkowski (1911)'s fundamental theorem on the geometry of numbers. The integer programming algorithm depends on algorithms we devise for finding in a similar time bound, the shortest nonzero vector in a lattice and the closest point of a lattice to a point in space. These then yield better algorithms than known before for simultaneous diophantine approximations (Lagarias (1982)).

While this paper presents mainly the theoretical improvements that can be made in the algorithms, we discuss in section 6 why in practice our estimates of running time may be overly pessimistic.

The last part of the paper discusses some complexity issues. It is an interesting open problem as to whether finding the Euclidean shortest non-zero vector of a given lattice is NP-hard. (See Lenstra (1981), Van Emde Boas (1981) and Lagarias (1982)).

We show first that this problem is polynomial-time Cook (Turing) reducible to the language:

SHORT=$\{(K;b_1,b_2,\dots,b_n) \mid b_1,\dots,b_n$ are integer n vectors and the Euclidean shortest, non-zero vector of the form $\sum_{i=1}^{n} z_i b_i$, z_i integers has length at most $k\}$

Using this reduction, we show that given a subroutine for SHORT which works in time T-SHORT (ℓ) -ℓ length of data, we can find in time ($p(n) \cdot$T-SHORT(ℓ)) (p-a poly) a good approximation to the closest vector of a point to a lattice. We conjecture that the latter problem is NP-Hard (see also conjecture 2 of Lagarias (1982)). The proof of this conjecture would say that SHORT is Cook NP-complete and our reduction is essentially a polynomial-time Turing rather than a Karp (many one) reduction. To my knowledge, no other language is known to be NP-complete under Turing reductions which is not trivially also many-one NP complete.

Notation

$\mathbb{R}^n$	-	Euclidean n space
$\mathbb{Z}^n$	-	set of n-vectors with integer components.

For vectors a and b (a,b) is the dot product of a and b.

for a vector a $|a| = |a|_2$ = Euclidean length of a

$|a|_1 = L_1$-length of a $= \sum_i |a_i|$

$L(b_1,b_2,\ldots,b_n)$ = lattice generated by vectors b_1, $b_2,\ldots,b_n$ which are possibly dependent.

If $b_1,b_2,\ldots,b_n$ is a list of vectors, we denote by $b_i(j)$ the projection of b_i onto the orthogonal complement of the space spanned by $b_1,b_2,\ldots,b_{j-1}$ for $i \geq j \geq 2$. $b_i(1)$ is defined to be equal to b_i for all i.

In each section, equations, inequalities, etc. are numbered successively beginning with 1. In the section in which it appears we refer to equation (10) - say simply as (10). But in other sections, we precede it with the section number. Thus (2.10) appearing in section 4 refers to (10) of section 2.

Section 1. Basic definitions and techniques.

A lattice in $\mathbb{R}^n$ (Euclidean n-space) is a set L generated by finitely many vectors $b_1,b_2,\ldots,b_k$ of $\mathbb{R}^n$, $L = \{\sum_{i=1}^{k} z_i b_i | z_i$ are all integers$\}$. If $b_1,\ldots,b_k$ are also independent, then we call them a basis of the lattice. It is a classical theorem that every lattice has a basis (see Cassels (1959) or Lekherkerker (1969)- these two books on geometry of numbers cover the subject extensively). If $b_1,b_2,\ldots,b_n$ are the basis of a lattice L in $\mathbb{R}^n$, then volume of the parallel piped enclosed by $b_1,b_2,\ldots,b_n$ is called the determinant of the lattice L = $L(b_1,b_2,\ldots,b_n)$ denoted d(L). This determinant is invariant under change of basis. It equals, of course, the determinant of the matrix which has $b_1,b_2,\ldots,b_n$ for its columns. The lattice generated by $b_1,b_2,\ldots,b_n$ remains invariant under the so called unimodular operations:

(1) Adding an integer multiple of one of the generators to another

(2) Multiplying a generator by -1.

Conversely it can be shown that if $b_1,b_2,\ldots,b_n$ and $\hat{b}_1,\hat{b}_2,\ldots,\hat{b}_n$ are two bases for the same lattice L, then we can get one from the other by a sequence of unimodular operations. Thus the number of elements in any basis of a lattice L is the same, this is called the dimension of the lattice.

We will have occasion to use the following fact often.

Proposition 1: Suppose v is an element of a lattice L, $v \neq 0$ and $\lambda v \notin L$ for $0<\lambda<1$. Then there is a basis of the lattice containing v.

Proof: (See for example Lekherkerker (1969) or Cassels (1959)). Let n be the dimension of the lattice. The proof will be by induction on n. For n=1, it can be checked that any lattice must be of the form $\{zb_1$: z an integer$\}$ and thus the only v that satisfies the hypothesis of the proposition is $v=b_1$. Now assume $n \geq 2$. Let $\hat{L}$ be the projection of L perpendicular to v, i.e.,

$$\hat{L} = \{b - \frac{(b,v)}{(v,v)} v: \ b \in L\}$$

(The quantity inside the parenthesis is the projection of b onto the hyperplane through the origin perpendicular to v). It is not difficult to see that $\hat{L}$ has dimension (n-1), let $\hat{b}_2$, $\hat{b}_3,\ldots,\hat{b}_n$ be a basis of $\hat{L}$. Let $b_2,b_3,\ldots,b_n$ be the vectors in L whose projections are $\hat{b}_2$, $\hat{b}_3,\ldots,\hat{b}_n$. We wish to assert that $\{v,b_2,\ldots,b_n\}$ is a basis of L. Clearly $v,b_2,\ldots,b_n$ are in L. It suffices to show that for any w in L, w can be expressed as $\sum_{i=1}^{n} z_i b_i$ with $v=b_1$, $z_i \in \mathbb{Z}$. Let

Let $\hat{w} = w - \frac{(w,v)}{(v,v)} v$. $\hat{w}$ is in $\hat{L}$ and hence equals $\sum_{i=2}^{n} z_i \hat{b}_i$, $z_i \in \mathbb{Z}$. Consider $(w - \sum_{i=2}^{n} z_i b_i) = w'$. w' must be a scaler multiple of v - say $= \lambda v$. If λ is not an integer, then since $w_1 = \lambda v$ and v are in L, $(\lambda - \lfloor\lambda\rfloor)$ v is in L contradicting the hypothesis. Hence λ is an integer and w = $\sum_{i=2}^{n} z_i b_i + \lambda v$ puts w in $L(v,b_2,\ldots,b_n)$. Note that the proof of the proposition actually proves the following stronger result:

Proposition 2: The following "algorithm" yields a basis $b_1,b_2,\ldots,b_n$ of the lattice L.

```
Procedure input lattice L of dimension n.  $b_0 \leftarrow 0$.
do for i = 1 to n by 1:
    Pick any $v \neq 0$ such that $(v,b_j)=0$ for j=0, 1,...,i-1.
    Find the smallest positive $\lambda$ such that $\lambda v$ is
    in the lattice $\hat{L}$ obtained by projecting L
    onto the orthogonal complement of span
    $\{b_1,\ldots,b_{i-1}\}$
    Find w in L such that w projects onto $\lambda v$ in $\hat{L}$.
    $b_{i+1} \leftarrow w$
end
return $\{b_1,b_2,\ldots,b_n\}$
```

Of course this is not quite an algorithm - we do not know how the input is specified etc. etc. But a more rigorous version of this algorithm will be given later - in section 2 - it is called SELECT-BASIS.

We are often interested in this paper in projecting and "unprojecting" vectors. Projecting a vector b onto a hyperplane through the origin with v as normal yields the vector $b - \frac{(b,v)}{(v,v)} v$. To project a vector onto a subspace, we just find an orthogonal basis of the complement of the subspace and project perpendicular to each basis vector successively. We also use one other observation. Suppose $b_1, b_2, \ldots, b_n$ form a basis of a lattice L and $\bar{L}$ is obtained by projecting $\underline{L}$ perpendicular to b_1. Then for any vector $\bar{w}$ in $\bar{L}$, there is a unique vector w in L such that $(w,b_1) \in [-\frac{|(b_1,b_1)|}{2}, \frac{|(b_1,b_1)|}{2})$ and $\bar{w}$ is w's projection perpendicular to b_1. To see this, let w' be some vector of L such that w' projects onto $\bar{w}$. Then $\{w'+z\ b_1: \ z \in \mathbb{Z}\}$ is contained in L and also has this property. Clearly a unique element of this set satisfies the dot product condition above. The set does not depend on the choice of w' and hence our assertion follows.

One other theorem we need from Geometry of Numbers is the fundamental theorem of Minkowski (1911).

Theorem 1. (Minkowski): If L is an n-dimensional lattice, the length (in Euclidean norm*) of a shortest nonzero vector of L is at most $\sqrt{n}(d(L))^{1/n}$.

This is a direct consequence of the more usual statement of Minkowski's theorem. See Lekerkerker (1969), Theorem 1, page 120 for example.

Preamble to the paper: Lenstra (1981) showed that given vectors $b_1, b_2, \ldots, b_n$ with integer coordinates in $\mathbb{R}^n$, one could find a basis $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ for the lattice $L(b_1, b_2, \ldots, b_n) = L$ such that

$$(1) \quad \prod_{i=1}^{n} |\hat{b}_i| \leq d(L) \cdot 2^{(n^2/4)} \quad .$$

He called bases satisfying the above condition "reduced" bases. His algorithm for doing this takes time $O(c^{n^3} \cdot \text{polynomial (length of data)})$. Later Lenstra, Lenstra and Lovasz (1982) gave a polynomial time algorithm for finding a reduced basis $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ and in addition to (1) showed for their reduced basis that

$$(2) \quad |\hat{b}_1| \leq 2^{n/2} \text{ . length of a shortest nonzero vector of L}$$

We will use the Lenstra, Lenstra, Lovasz algorithm in our algorithm. So familiarity with these two very important papers, especially the basis reduction algorithm of the latter is highly recommended for reading this paper. However, we have used their basis reduction algorithm only as a black box and this paper should be quite self-contained if the reader remembers that the LLL algorithm yields a basis satisfying (2) above. The one result from Lenstra (1981) that we use, again, only as a black box in section 5 may be stated as follows: Given a polytope $\{x: Ax \leq b\}$ in $\mathbb{R}^n$ where A and b are the data, we can find in polynomial time a linear transformation τ, p in $\mathbb{R}^n$ in $\mathbb{R}^n$ and R, r $\in \mathbb{R}$ SUCH that

$$B(p,r) \subseteq \{\tau x: Ax \leq b\} \subseteq B(p,R)$$

and $\quad R/r \leq 2n^{3/2}$

where B(y,s) is the ball of radius s with y as centre. This says, roughly, that we can apply a linear transformation to make the polytope "round."

Section 2: The algorithm for finding the shortest vector

A. First we describe our algorithm to find the shortest nonzero* vector in a lattice L given by its generators, $b_1, b_2, \ldots, b_n$ and then show how this is used for integer programming. The procedure that finds a shortest nonzero vector in n dimensions works recursively by calling the subroutine for lattices of dimension (n-1) or less. Using polynomially many calls on such lower dimentional subroutines, we find a basis $a_1, a_2, \ldots a_n$ for lattice $L(b_1, \ldots, b_n)$ $(b_1, \ldots, b_n$ given) which satisfies the following properties:

(1) for j=2,...,n-1, $a_j(j)$ is the shortest vector in the lattice L_j generated by $a_j(j), a_{j+1}(j), \ldots, a_n(j)$ which is (n-j+1) dimensional.

$$(2) \quad |a_1| \leq \frac{2}{\sqrt{3}} |a_2|$$

$$(3) \quad |a_2 - a_2(2)| \leq \left|\frac{a_1}{2}\right|$$

Whereas in the reduced basis of Lenstra, Lenstra and Lovasz (1982), the length of the first vector is guaranteed to be at most $2^{(n/2)}(d(L))^{1/n}$, for our basis $a_1, \ldots, a_n$, we can easily prove (4) below using Minkowski's theorem, conditions (2) (3) and the fact that $d(L) = |a_1| d(L_2)$

$$|a_1| \leq \sqrt{2n}\ (d(L))^{1/n}$$

(In other words, our a_1 is a much shorter vector in general than theirs - but of course we spend more time finding it).

Having obtained such a basis $a_1, \ldots, a_n$, we show that the shortest vector must be of the form

* Throughout the paper, unless stated otherwise we use the L_2 norm.

* We sometimes use the phrase "shortest vector" for "shortest nonzero vector" when the meaning is clear

$$y = \sum_{i=1}^{n} \alpha_i a_i \quad \text{where } (\alpha_1,\dots,\alpha_n) \in T$$

where T is a subset of $\mathbb{Z}^n$. We show that it is enough to consider a set T of cardinality at most

$$\frac{2^n |a_1|^n}{d(L)} \tag{5}$$

(4) is used to bound the expression (5). We enumerate all members of T, find the corresponding y and take the shortest.

We now present an algorithm that finds a shortest nonzero vector in the lattice $L(b_1,\dots,b_n)$ where $b_1,\dots,b_n$ are independent input vectors with integer coordinates. To facilitate the recursion, the algorithm will accomplish more. To describe what more, suppose $v_1,v_2,\dots,v_n$ is the final basis returned by the algorithm. As before let $v_i(j)$ denote the projection of v_i onto the (n-j) dimensional subspace V_j of $\mathbb{R}^n$ orthogonal to the span of $v_1,v_2,\dots,v_{j-1}$, for $i \ge j \ge 2$. Let $v_1(1) = v_1$ and $V_1 = \mathbb{R}^n$. Then our basis $\{v_1,\dots,v_n\}$ satisfies: (cf: (1) through (3)).

(6) for $j=1,2,3,\dots,n-1$, $v_j(j)$ is the shortest vector in L_j - the projection of the lattice $L(b_1,\dots,b_n) = L$ onto V_j. (Thus v_1 is the shortest vector in L).

(7) $|v_i(j) - v_i(j+1)| \le \frac{|v_j(j)|}{2}$ for $i \ge j \ge 1$

Procedure. SHORTEST $(n;b_1,b_2,\dots,b_n)$

Comment. The preceding paragraph explains what exactly this procedure accomplishes. $L = L(b_1,\dots,b_n)$.

```
1.  If n=1 then return $(b_1)$.
2.  Use the basis reduction algorithm of Lenstra, Lenstra and Lovasz (1982) to make
    $\{b_1,b_2,\dots,b_n\}$ a reduced basis.
3.  $\bar{b}_i$ ← projection of $b_i$ perpendicular to $b_1$
      for $i=2,3,\dots,n$.
4.  $\{\bar{b}_2,\bar{b}_3,\dots,\bar{b}_n\}$ ← SHORTEST $(n-1;\ \bar{b}_2,\bar{b}_3,\dots \bar{b}_n)$.
5.  for i=2 to n do
        Find the unique element $a_i$ in L such that
        $a_i$'s projection perpendicular to $b_1$ is
        $\bar{b}_i$ and $(a_i,b_1) \in (-\frac{|(b_1,b_1)|}{2}, \frac{|b_1,b_1)|}{2}]$
        $b_i$ ← $a_i$
    end
```

Comment. We now have a basis $b_1,b_2,\dots,b_n$ satisfying conditions (1) and (3).

```
(6) If $|b_2| < \sqrt{\frac{3}{2}}|b_1|$ then do  Swap $b_1$ and $b_2$
                                                  go to 3.
    end
```

Comment. We now satisfy condition (2); caution: $|b_2|$, $|b_1|$ may be irrational, but their squares are not, so we use them instead.

```
(7)  If $|b_j(j)| \ge |b_1|$ for some j, then
         $j_0$ ← minimum such j else
         $j_0$ ← n+1
(8)  BASIS ← $\{b_1,b_2,\dots,b_{j_0-1}\}$ .
```

Comment. We show later that some nonzero shortest vector of $L(b_1,\dots,b_n)$ is in $L(b_1,\dots,b_{j_0-1})$.

```
(9)  Call ENUMERATE(BASIS) to obtain $v_1$ = a
     shortest nonzero vector in $L(b_1,\dots,b_n)$.
```

Comment. This procedure is explainer later.

```
(10) $\{b_1,b_2,\dots,b_n\}$ ← SELECT_BASIS$(n;v_1, b_1,\dots,b_n)$
```

Comment. Procedure explained later. It returns a basis of L containing v_1 as the first vector (cf. Proposition 1.1)

```
(11) $\bar{b}_i$ ← projection of $b_i$ perpendicular to
     $b_1$ for $i=2,\dots,n$
(12) $\{\bar{b}_2,\bar{b}_3,\dots,\bar{b}_n\}$ ← SHORTEST $(n-1;\ \bar{b}_2, \bar{b}_3,\dots,\bar{b}_n)$
(13) For $i=2,3\dots,n$ find $a_i$ such that $a_i$ is
     in L, $a_i$'s projection perpendicular to
     $b_1$ is $\bar{b}_i$ and $(a_i,b_1) \in (-\frac{|(b_1,b_1)|}{2}, \frac{|(b_1,b_1)|}{2}]$
(14) Return $(b_1,a_2,\dots,a_n)$.
```

Procedure. SELECT-BASIS$(n;b_1,b_2,\dots,b_{n+1})$

Comment. $b_1,b_2,\dots,b_{n+1}$ are rational vectors in $\mathbb{R}^k$ for some $k \ge n$ and span an n-dimensional subspace of $\mathbb{R}^k$. The procedure returns a basis $a_1,a_2,\dots,a_n$ of $L=L(b_1,b_2,\dots b_{n+1})$. It first finds the shortest vector of L in the direction of b_1 - this is a_1. Then it projects L orthogonal to a_1 to get a lattice $\bar{L}$. It works by recursively finding a basis of $\bar{L}$ (cf. Proposition 1.2).

```
If n=0 then return the empty set.
If $b_1$ is independent of $b_2,b_3,\dots,b_{n+1}$
   then $a_1 \leftarrow b_1$
   else do
        find $\alpha_2,\alpha_3,\dots,\alpha_{n+1}$ (rationals-these are
        unique) such that $\sum_{j=2}^{n+1} \alpha_j b_j = b_1$
        M least common multiple of the denominators
        of $\alpha_2,\dots,\alpha_{n+1}$
        $\gamma \leftarrow \gcd(M\alpha_2, M\alpha_3,\dots,M\alpha_{n+1})$
          if $(M/\alpha)$ is integral then $a_1 \leftarrow b_1$
                     else do:
                     find p, $q \in \mathbb{Z}$ rel-prime
                     such that $\frac{M}{\gamma} - \lfloor \frac{M}{\gamma} \rfloor = p/q$
                     $a_1 \leftarrow \frac{1}{q} b_1$

                     end
Let $\bar{b}_2,\bar{b}_3,\dots,\bar{b}_{n+1}$ be the projections of
$b_2,\dots,b_{n+1}$ perpendicular to $a_1$; if any one of
$\bar{b}_2,\dots,\bar{b}_{n+1}$-say $\bar{b}_j$ is zero then delete it to get
a basis $\bar{a}_2,\dots,\bar{a}_n$ for $L(\bar{b}_2,\bar{b}_3,\dots,\bar{b}_{n+1})$.
else Call SELECT_BASIS $(n-1;\bar{b}_2,\bar{b}_3,\dots,\bar{b}_{n+1})$ to
get $\bar{a}_2,\dots,\bar{a}_n$ as the basis of $L(\bar{b}_2,\bar{b}_3,\dots,\bar{b}_{n+1})$.
Find $a_2,a_3,\dots,a_n$ in L such that $a_j$ projects onto
$\bar{a}_j$ and $|(a_j,a_1)| \leq |(a_1,a_1)|/2$
Return $(a_1,a_2,\dots,a_n)$
end.
```

Proposition 0. The basis $a_1,a_2,\dots,a_n$ returned by the above procedure is a basis of $L = L(b_1,b_2,\dots,b_{n+1})$ assuming $b_1,b_2,\dots,b_{n+1}$ have rank n.

Proof: By Proposition 1.2, it suffices to show that a_1 is the shortest vector of L in the direction b_1. We have

$$M\, b_1 = \gamma \left[\frac{M\alpha_2}{\gamma} b_2 + \frac{M\alpha_3}{\gamma} b_3 + \dots + \frac{M\alpha_{n+1}}{\gamma} b_{n+1} \right] \quad \text{and}$$

$\frac{M\alpha_2}{\gamma}, \dots, \frac{M\alpha_{n+1}}{\gamma}$ are relatively prime integers. Any other vector in $L(b_2,\dots,b_{n+1})$ in the direction of b_1 must thus be an integer multiple of the vector in the square brackets which equals $\frac{M}{\gamma} b_1 = \frac{p}{q} b_1$. Thus any vector of $L(b_1,b_2,\dots,b_{n+1})$ in the direction of b_1 is an integer multiple of $\frac{1}{q} b_1$. Conversely, $\frac{p}{q} b_1$ and $\frac{q}{q} b_1$ belong to L and p and q are relatively prime implies $\frac{1}{q} b_1$ does too. Hence we have the lemma.

Proposition 1. The vectors $v_1,\dots,v_n$ returned by the procedure SHORTEST $(n;b_1,b_2,\dots,b_n)$ form a basis of $L(b_1,\dots,b_n)$.

Proof: For n=1, the proof is clear. We proceed by induction. At the end of step 4 of the procedure, $\bar{b}_2,\bar{b}_3,\dots,\bar{b}_n$ form a basis of the lattice $L_2=L$ projected onto the hyperplane h perpendicular to b_1 at the origin and thus by Proposition 1.2, $a_1,a_2,\dots,a_n$ form a basis of $L(b_1,\dots,b_n)$ at the end of step 5. By repeating this argument, they form a basis of L at the beginning of step 10. By Proposition 0, again procedure SELECT-BASIS works correctly to produce a basis of our lattice. Hence the current proposition is proved.

Proposition 2. The vectors returned by SHORTEST satisfy conditions (6) and (7).

Proof: Easy.

Proposition 3. Let j_0 be as defined in step 7 of procedure SHORTEST. Then a shortest nonzero integer combination of $b_1,b_2,\dots,b_{j_0-1}$ is also a shortest nonzero integer combination of b_1, $b_2,\dots,b_n$.

Proof: Suppose $v = \sum_{i=1}^{n} \alpha_i b_i$ is a shortest nonzero integer combination of $b_1,b_2,\dots,b_n$ and one of $\alpha_{j_0}, \alpha_{j_0+1},\dots, \alpha_n$ is nonzero. Then the projection v' of v onto V the orthogonal complement of span $\{b_1,b_2,\dots,b_{j_0-1}\}$ is nonzero. But then since the shortest vector of L projected onto V is of length $b_{j_0}(j_0)$, (by (6)) $|v'| \geq |b_{j_0}(j_0)|$. Then clearly $|v| \geq |v'| \geq |b_{j_0}(j_0)| \geq |b_1|$. Thus b_1 is a shortest vector of L. This proves proposition 3.

Proposition 4. The procedure SHORTEST calls itself on lower dimensional lattices at most $\frac{5}{2}$ n times (when started on a n dimensional lattice).

Proof: By Lenstra, Lenstra and Lovasz, the execution of their basis reduction algorithm (in step 2 of procedure SHORTEST)-yields a basis of L with $|b_1| \leq 2^{(\frac{n-1}{2})} \lambda_1(L)$ where $\lambda_1(L)$ is the length of the shortest vector of L. Each execution, but the first of the loop of steps 3-6 of SHORTEST cuts down $|b_1|$ by a factor of $(\frac{\sqrt{3}}{2})$. Thus each 5 iterations of the loop cuts it down by a factor of 2. Thus there are at most (5 n/2) executions of the loop.

Description of procedure ENUMERATE. Suppose BASIS = $\{b_1, b_2, \ldots, b_m\}$ in step 8 of procedure SHORTEST and suppose a shortest vector of $L(b_1, b_2, \ldots, b_m)$ is $y = \sum_{i=1}^{m} \alpha_i b_i$. Then since y must be of length at most $|b_1|$ and the projection of y onto V_m the orthogonal complement in $\mathbb{R}^n$ of span $\{b_1, \ldots, b_{m-1}\}$ must be of length at most $|b_1|$. This projection has length $|\alpha_m b_m(m)|$, we must have

$$|\alpha_m| \leqq \frac{|b_1|}{|b_m(m)|} .$$

More generally we have the following lemma:

Proposition 5. With the above notation, suppose $\beta_{i+1}, \beta_{i+2}, \ldots, \beta_m$ are fixed integers. Then there is an easily computed integer β_i^0 such that for all integers $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}$ and β_i,

$$\left| \sum_{j=1}^{i-1} \alpha_j b_j + \beta_i b_i + \sum_{j=i+1}^{m} \beta_j b_j \right| \leqq |b_1|$$

$$\beta_i \varepsilon \left[\beta_i^0,\ \beta_i^0 + 2\,\frac{|b_1|}{|b_i(i)|}\right]$$

Proof: For any vector v, we denote by $\hat{v}$ the component of v along $b_i(i)$, i.e., $\frac{(v, (b_i(i))}{(b_i(i), b_i(i))} b_i(i)$. Let $u = \sum_{j=i+1}^{m} \beta_j b_j$ and $w = \sum_{j=1}^{i-1} \alpha_j b_j + \beta_i b_i + \sum_{j=i+1}^{m} \beta_j b_j$. Clearly, $\hat{w} =$

$\beta_i b_i(i) + \sum_{j=i+1}^{m} \beta_j \hat{b}_j$. Since β_j, $j=i+1, \ldots, m$ are fixed, the last sum is fixed. Let it be $(t\, b_i(i))$, t a scaler. Then

$$\hat{w} = \beta_i b_i(i) + t\, b_i(i) = (\beta_i + t) b_i(i)$$

$$|w| \leqq |b_1| \rightarrow |\hat{w}| \leqq |b_1| .$$

$\{(\beta_i + t) b_i(i) \mid \beta_i \varepsilon \mathbb{Z}\}$ are colinear points at distance $|b_i(i)|$ between two adjacent ones. Clearly then the lemma follows.

NOTE: We needed $|b_1| \leqq |b_i(i)|$ to state the lemma as it is. If $|b_1| < |b_i(i)|$, we need to try at most 2 values of α_i - not $2\,(|b_1|/|b_i(i)|)$.

```
Procedure ENUMERATE $(b_1, b_2, \ldots, b_m)$
Comment. $b_1, b_2, \ldots, b_m$ are vectors satisfying
(1), (2) and (3). The procedure here finds the
shortest vector of $L(b_1, b_2, \ldots, b_m)$.
if m=1 then return $(b_1)$.
for each $\alpha_m$ integer in the range
$-\frac{|b_1|}{|b_m(m)|}$ to $+\frac{|b_1|}{|b_m(m)|}$
do:
    CALL LIST (m-1)
end:
Comment: LIST(m-1) returns T - a list of candi-
dates $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ as per proposition 5.
Return the shortest nonzero vector in the set
$\{\sum_{j=1}^{m} \alpha_j b_j :\ (\alpha_1, \ldots, \alpha_n) \varepsilon T\}$.
end ENUMERATE.

procedure LIST(k)
Comment. When this procedure is called, the
integers $\alpha_{k+1}, \alpha_{k+2}, \ldots, \alpha_m$ are fixed.
if k=0 then do
        $T \leftarrow T \cup \{(\alpha_1, \alpha_2, \ldots, \alpha_m)\}$
        Return
    end
Compute $\beta_k^0$ based on proposition 5.
for each integer $\alpha_k$ in the range
$\left[\beta_k^0,\ \beta_k^0 + 2\frac{|b_1|}{|b_k(k)|}\right]$
 do
    Call LIST(k-1)
end
end LIST.
```

Lemma 1. The procedure ENUMERATE tries out at most $2^m \prod_{i=2}^{m} (|b_1|/b_i(i)|))$ sets of values which is at most $(2n)^{n/2}$ sets of values.

Proof: The first part follows from the last proposition. The denominator $\prod_{i=2}^{m} |b_i(i)|$ is clearly the determinant of the lattice L' generated by $b_2(2), b_3(2), \ldots, b_m(2)$. $|b_2(2)|$ is the length of the shortest vector in L', thus

$$\frac{|b_2(2)|^{m-1}}{d(L')} \leqq (\sqrt{m-1})^{m-1} \leqq n^{n/2} \text{ (by Minkowski).}$$

Further $|b_1| \leqq \frac{2}{\sqrt{3}} |b_2|$ and $|b_2 - b_2(2)| \leqq \frac{|b_1|}{2}$ imply that $(\sqrt{2}|b_2(2)|) \geqq |b_1|$. Thus we need to enumerate at most

$$(\sqrt{2})^{(m-1)} \cdot n^{n/2} \leqq (2n)^{n/2} \text{ possibilities.}$$

We show in the next section that if the input vectors to SHORTEST (n;...) are all integral and each is of length at most B, then all numbers produced by the algorithm are rationals with numerators and denominators that can be represented in $O(n^2(\log B + \log n))$ bits. (lemma 3 of section 3). Assuming this for the moment, we prove the following theorem.

Theorem 1: With input vectors of integer components each of length at most B, the algorithm SHORTEST(n;...) runs in time $O((4n^{3/2})^n(\log B)(\log\log B))$.

NOTE: In common usage, we might call this a $O((4n^{3/2})^n)$ algorithm. This however counts only the number of arithmetic operations-additions, substractions, multiplications and divisions and not the number of bit operations. Since log B could be substantial, and the number of bits of numbers grows nontrivially, such an analysis is imprecise.

Proof: Let T(n) be the maximum number of arithmetic operations (not bit operations) performed by SHORTEST(n;...) for any input $b_1,\ldots,b_n$ of n vectors. Then it is easily seen that all steps of the algorithm except recursive calls to SHORTEST(n-1;....) and the enumeration step take only polynomially many operations. Thus we have,

$$T(n) \leqq \frac{5n}{2} T(n-1) + q(n) \quad \text{q-some polynomial}$$
$$+ (2n)^{n/2}$$

(by proposition 4 and lemma 1)

Thus T(n) is $O((3\cdot 9)n^{3/2})^n)$. By lemma 3 of the next section, each arithmetic operation is performed on operands of length $O(n^2(\log B + \log n))$ and hence take $O(n^2(\log B + \log n)\cdot(2\log n + \log\log B + \log\log n))$ bit operations. This multiplied by T(n) is at most $O((4n^{3/2})^n \cdot \log B \log\log B)$. Hence the theorem.

Section 3. Size of the numbers involved in the algorithm.

Since this algorithm manipulates numbers and keeps all numbers exactly, it is important to derive bounds on the number of bits needed to represent them. It is easy to see that they are rationals. We will derive bounds on the size of the numberators and denominators of all these numbers.

First we note that even though the algorithm at various times works on the projections of the original lattice L; it always has values of $b_1(1)$, $b_2(2),\ldots,b_n(n)$ for some basis $b_1,\ldots,b_n$ of the whole lattice. Here as usual $b_i(i)$=projection of b_i onto the orthogonal complement of the space spanned by $b_1,\ldots,b_{i-1}$ for $i \geqq 2$ and $b_1(1) = b_1$.

Proposition 1. $\max_{i=1}^{n} |b_i(i)|$ never increases during the execution of SHORTEST.

Proof: We consider the algorithm step by step. The proof is by induction on n. For n=1, the proof is trivial. So assume $n \geqq 2$. For step 2, the LLL algorithm never increases the quantity as seen from their paper (page 13, last but one paragraph). For step 4, the inductive hypothesis suffices. In step 6, $|b_1(1)|$ strictly decreases, $|b_2(2)|$ does not increase and $|b_3(3)|,\ldots, |b_n(n)|$ remain the same. For steps 7 through 10, the enumeration and basis selection processes, the proof is a little harder, but is dealt with in proposition 2. For step 13 again, we invoke the inductive hypothesis, completing the proof of this proposition.

Proposition 2. Steps 7 through 10 of the algorithm SHORTEST $(n;b_1,\ldots,b_n)$ do not increase $\max_i |b_i(i)|$.

Proof: Suppose $b_1,\ldots,b_n$ is the basis of the lattice at the beginning of step 7 and suppose v_1 is found to be a shortest nonzero vector of $L(b_1,\ldots,b_n)$ by enumeration. Define $u_1=v_1$, $u_2=b_1$, $u_3=b_2\ldots$ $u_{n+1}=b_n$. Let $u_i(j)$ be defined as usual for $i \geqq j$. Clearly, precisely one of the $u_i(i)$'s is zero. Let this be $u_j(j)$. Finally let $\hat{b}_1,\hat{b}_2,\ldots,\hat{b}_n$ be the basis returned by SELECT_BASIS$(v_1,b_1,\ldots,b_n)$. $\hat{b}_1=v_1$. Further for $l=2,3,\ldots,j-1$, the algorithm SELECT_BASIS chooses $\hat{b}_l(l)$ to be a vector of length at most $|u_\ell(\ell)|$. Now $(u_\ell(\ell))$=projection of $b_{\ell-1}$ orthogonal to $v_1,b_1,\ldots,b_{\ell-2}$ and thus we must have

(1) $|\hat{b}_\ell(\ell)| \leqq |u_\ell(\ell)| \leqq |b_{\ell-1}(\ell-1)|$ for $\ell=2,3,\ldots,j-1$. For $\ell=j,j+1,\ldots,n$ $|\hat{b}_\ell(\ell)|$ is the same as $|b_\ell(\ell)|$ because $b_{j-1}(j-1)$ is discarded and after that SELECT_BASIS just returns what is left.

(2) $|\hat{b}_\ell(\ell)| = |b_\ell(\ell)|$ for $\ell=j,j+1,\ldots,n$ and

(3) $|\hat{b}_1(1)| = |v_1| \leqq |b_1(1)|$. (1), (2), and (3) establish the proposition.

We now define, for any basis $b_1,\ldots,b_n$ of the n-dimensional lattice

(4) $d_i = (d(L(b_1,b_2,\ldots,b_i)))^2$.

It is easy to see that d_i is the determinant of the i×i matrix with entries (b_j,b_l) for $1 \leqq j$, $l \leqq i$. Since our original basis vectors had integer coordinates, this is also true of any other basis. Thus the d_i are all integers. Clearly,

$$(5) \qquad d_i = \prod_{j=1}^{i} |b_j(j)|^2$$

The following proposition resembles a similar one in the LLL paper.

Proposition 3. All numbers produced by the algorithm are rationals of the form p/q, p,q in $\mathbb{Z}$ where q is one of the d_i's.

Proof: $b_j(i)$, the projection of b_j orthogonal to $b_1,\ldots,b_{i-1}$ (for $j \geq i \geq 2$) is given by (6) $b_j(i) = b_j - \sum_{k=1}^{i-1} \delta_{jk} b_k$ where δ_{jk} are some real numbers. Taking a dot product with b_ℓ $(1 \leq \ell \leq i-1)$ and noting that $(b_\ell, b_j(i)) = 0$, we have

$$(b_j, b_\ell) = \sum_{k=1}^{i-1} \delta_{jk}(b_k, b_\ell) \text{ for } \ell = 1,2,\ldots,i-1.$$

These are $(i-1)$ independent equations in the $(i-1)$ variables δ_{jk} with a matrix whose determinant is d_{i-1}. Thus $(d_{i-1}\delta_{jk})$ are all integers. Hence from (6) $(b_j(i)d_{i-1})$ is an integral vector. The proof of the proposition for other quantities - for example the μ_{ij}'s of the LLL algorithm is similar and we omit it.

Lemma 1: Except while executing the Lenstra, Lenstra and Lovasz algorithm and step 9,10 all the vectors produced by the algorithm SHORTEST $(n;b_1,\ldots,b_n)$ are of length at most $nB^{1/2}$ where the input $b_1,\ldots,b_n$ consists of integral vectors each of length at most $\sqrt{B}$.

Proof: While $|b_1(1)|$ never increases, $|b_i(i)|$ could increase for $i>1$. So to prove the lemma, we proceed by induction, but to avoid mistakes one must prove by induction on i that SHORTEST $(i;a_1,a_2,\ldots,a_i)$ never produces vectors that are too long where i is between 1 and n and $a_j = b_{n-i+j}(n-i+j)$ for $j=1,2,\ldots,i$. Throughout this proof $\sqrt{B}$ will denote the length of the longest of $b_1,\ldots,b_n$ which were the original input to SHORTEST$(n;\ldots)$. For $i=1$, the proof is trivial. Assume $i \geq 2$ and no execution of SHORTEST$(i-1;\ldots)$ within SHORTEST$(n;\ldots)$ ever produces a vector of length more than $(i-1)\sqrt{B}$ except while executing LLL. We consider an execution of SHORTEST$(i;a_1,\ldots,a_i)$ where again $a_j = b_{n-i+j}(n-i+j)$ for $j=1,2,\ldots,i$ step by step.

Step 1 OK Step 2 exclused by hypothesis. Step 3 does not increase lengths of vectors. Step 4 Here SHORTEST$(i-1;\ldots)$ is called and we use induction to assert that the lengths of all vectors are bounded by $(i-1)\sqrt{B}$ except while executing LLL.

Step 5: This adds one more "component" to the vectors, but the component is no more than the current $|b_{n-i+1}(n-i+1)|/2$ which by Proposition 1 is at most $B^{1/2}$. Thus from the previous sentence, the lengths of the vectors at the end of Step 5 are at most $(i-1)\sqrt{B} + \sqrt{B} = i\sqrt{B}$. Steps 6,7 and 8 do not affect the lengths of the vectors. Steps 9 and 10 are excluded by hypothesis. But we note that at the end of SELECT_BASIS, we have a basis $a_1,a_2,\ldots,a_i$ such that

$$|a_\ell(\ell)| \leq \sqrt{B} \quad \text{(by Proposition 2)}$$

and each a_j can be expressed as

$$a_j = a_j(j) + \sum_{i=1}^{j-1} \mu_{ji} a_i(i) \quad \text{where } |\mu_{ji}| \leq \frac{1}{2} \text{ for all } i,j.$$

(To see this use induction on k where k is the argument if SELECT_BASIS-i.e., deal with SELECT_BASIS$(k;\ldots)$). Thus $|a_\ell| \leq i\sqrt{B}$ for all ℓ. The proof that steps 11,12 and 13 keep the bounds on the lengths is similar to that for steps 3,4 and 5 and is therefore omitted. This proves our current lemma. We will now consider the "problem" steps separately.

Lemma 2: In SHORTEST$(n;b_1,\ldots,b_n)$ during every execution of LLL algorithm all numbers produced are bounded in magnitude by $(nB)^{cn^2}$ for some fixed constant c.

Proof: Whenever the LLL algorithm is called, all the input vectors - say - $a_1,a_2,\ldots,a_i$ have rational components with common denominator d where by Proposition 3, d is one of the d_i and hence by Proposition 2 is bounded by B^i. Also, by the previous lemma, their lengths are all bounded by $(n\sqrt{B})$. Further, it is easily seen that the LLL algorithm behaves identically on input $(da_1, da_2,\ldots, da_i)$ as it does on input $a_1,a_2,\ldots,a_i$ except that in the second case all vectors are divided by d. $(da_1,\ldots,da_i)$ are integral vectors and thus the bounds proved in the LLL paper apply to them. For these as input, we have (from their Proposition (1.26)) that all numbers produced by their algorithm are bounded in magnitude by

$$(\max_{i=1}^{n} |da_i|)^{cn}$$

which is at most

$$(B^{n-1}\, n \cdot \sqrt{B})^{cn} \leq (nB)^{cn^2}$$

Proposition 4. Steps 9 and 10 do not produce any number of magnitude greater than $n^{cn}B$.

Proof: It is easily seen using Minkowski's theorem that in the enumeration process, all the multipliers of the vectors are at most n^{cn} for some constant . SELECT_BASIS is also easy to analyse. Hence the proposition follows.

Lemma 3: On input $b_1,\ldots,b_n$ which are independent vectors with integer components and of length at most $\sqrt{B}$, all numbers produced by the algorithm SHORTEST$(n;b_1,\ldots,b_n)$ can be represented in $O(n^2(\log B+\log n))$ bits. Further, the numbers produced by the algorithm can be represented in $O(n\log n+\log B)$ bits except while it is executing the Lenstra, Lenstra, Lovasz algorithm.

Section 4. Finding the closest vector:

In this section, we consider the following "closest vector problem":

Given $b_1,b_2,\ldots,b_n$ independent vectors in

(1) $\mathbb{R}^n$ with rational components and b_0 in $\mathbb{R}^n$, find b in $L(b_1,\ldots,b_n)$ such that $|b_0-b|$ is a minimum. This is the inhomogeneous problem. We solve this as a "warm-up" for the integer programming problem in the next section. The algorithm we give first finds using SHORTEST "a nice" basis $\hat{b}_1, \hat{b}_2,\ldots,\hat{b}_n$. (I.e., a basis satisfying (2.6) and (2.7)). Next we use an upper bound $\frac{1}{2}\sum_{j=1}^{n}|\hat{b}_j(j)|= M$ (say) on the distance between any b_0 and its closest lattice point. (This bound is proved in Proposition 1 to follow) to assert that we need to enumerate not too many values of $(\alpha_1,\alpha_2,\ldots,\alpha_n)$ such that $|\sum_{j=1}^{n}\alpha_j\hat{b}_j-b_0|$ is within this upper bound. Arguing as in the case of shortest vectors (Proposition 5 and lemma 1 of section 2), this gives us a bound of $M^n/d(L)$ for the number of possible n-tuples $(\alpha_1,\ldots,\alpha_n)$ to enumerate. Unfortunately, this can be too large. So we have to use another idea: If $|\hat{b}_i(i)|$ is the largest among all $|\hat{b}_j(j)|$, then we are able to show that not too many values of $(\alpha_i,\alpha_{i+1}\ldots,\alpha_n)$ are candidates to be tried. For each such candidate, we project to a (i-1) dimensional problem and solve these recursively. The details are explained after the algorithm.

```
procedure CLP$(n;b_0,b_1,\ldots,b_n)$

Comment: This procedure returns the vector in $L(b_1, b_2,\ldots,b_n)$ that is $L_2$-closest to $b_0$. $b_1,\ldots,b_n$ are
independent vectors in $\mathbb{Z}^k$, $k\geq n$.
$\{b_1,\ldots,b_n\} \leftarrow$ SHORTEST $(n;b_1,b_2,\ldots,b_n)$
Return CLP'$(n;b_0,b_1,\ldots,b_n)$
end CLP.

procedure CLP' $(n^0,b_0,b_1,\ldots,b_n)$

Comment: Does the same as CLP, but assumes $b_1,\ldots b_n$ is a "nice" basis for $L(b_1,\ldots,b_n)$-i.e., is the
output of SHORTEST.

if n=1 then return the easily computed closest point. Find i such that
$|b_i(i)| = \max_{j=1}^{n}|b_j(j)|$.

CANDIDATES $\leftarrow \emptyset$ .
for each "possible" $\lambda_i,\lambda_{i+1},\ldots,\lambda_n$ in $\mathbb{Z}$ do:
Comment: This is the enumeration step, explained later.
if i=1 then CANDIDATES$\leftarrow$CANDIDATES $\cup\{\sum_{j=i}^{n}\lambda_j b_j\}$
else do
    $v \leftarrow \sum_{j=i}^{n}\lambda_j b_j$

    $v' \leftarrow$ CLP'$(i-1;b_0-v, b_1,b_2,\ldots,b_{i-1})$

    CANDIDATES$\leftarrow$CANDIDATES $\cup\{v'+v\}$

end

Return the element of CANDIDATES that is closest to $b_0$.

end CLP'
```

To show that the number of $(\lambda_i,\lambda_{i+1},\ldots,\lambda_n)$ that we need to enumerate is small, we need to show the proposition.

Proposition 1. If $L=L(b_1,\ldots,b_n)$ is a lattice in $\mathbb{R}^k, k\geq n$ $b,\ldots,b_n$ independent and $b_0\in\mathbb{R}^k$ with $\bar{b}_0=$ projection of b_0 onto span $\{b_1,b_2,\ldots,b_n\}$, then there exists a point b in L such that

$$|b-\bar{b}_0|\leq\frac{1}{2}\sum_{j=1}^{n}|b_j(j)| .$$

Further if i is such that $|b_i(i)|=\max_j|b_j(j)|$, then clearly, $|b-\bar{b}_0|\leq|\frac{n}{2}b_i(i)|$.

Proof: It is not difficult to see that we can successively choose integers $\alpha_n,\alpha_{n-1},\ldots,\alpha_1$ (in that order) such that

$$\left|\left(\sum_{\ell=j}^{n}\alpha_\ell b_\ell-\bar{b}_0\right), b_j(j)\right| \leq \frac{|b_j(j)|}{2}$$

for all j. (Because the choice of α_j does not affect the earlier inequalities obtained). Since $b_1(1), b_2(2),\ldots,b_n(n)$ form an orthogonal basis of the space span $\{b_1,\ldots,b_n\}$ and $\bar{b}_0$ also lies in that space, the lemma follows.

Proposition 2. With the notation of the last proposition, there exists an easily determined set $T\subseteq\mathbb{Z}^{n-i+1}$ with $|T|\leq n^{\frac{3}{2}(n-i+1)}$ such that if $\sum_{j=1}^{n}\lambda_j b_j$ is a closest point to $b_0,\ldots,\lambda_j\in\mathbb{Z}$, then $\{\lambda_i,\lambda_{i+1},\ldots,\lambda_n\}$ belongs to T.

Proof: Suppose $\sum_1^n\lambda_j b_j=v$ in L is a closest point to b_0. Then clearly, v must be the closest point in L to $\bar{b}_0$. By the last proposition, then we must have $|v-\bar{b}_0| \leq \frac{n}{2}|b_i(i)|$. Now we have $|v-\bar{b}_0| \geq |((v-\bar{b}_0), b_n(n)|/|b_n(n)| = |\lambda_n||b_n(n)| - \frac{|(\bar{b}_0, b_n(n))|}{|b_n(n)|} = |\lambda_n||b_n(n)| - t$ for some fixed real number t. Thus these are at most $(\frac{n}{2}|b_i(i)|/|b_n(n)|)$ candidates for λ_n. Now we show a similar bound for $\lambda_i, \dots, \lambda_n$ using an argument very like that of Proposition 5 of Section 2. So suppose $\lambda_{j+1}, \lambda_{j+2}, \dots, \lambda_n$ are fixed integers, $j \geq i$. Then arguing as in that proposition, there are at most $\max(2, 2 \cdot \frac{n}{2}|b_i(i)|/|b_j(j)|) = n|b_i(i)|/|b_j(j)|$ possible values of λ_j such that the length of v in the direction of $b_j(j)$ remains bounded by $\frac{n}{2}|b_i(i)|$. Clearly as before we may easily compute these values λ_j. Thus we have shown that we need to consider a set T of candidates $\{\lambda_i, \lambda_{i+1}, \dots, \lambda_n)$ where

$$|T| \leq \prod_{j=i}^{n} n|b_i(i)|/|b_j(j)|$$

$|b_i(i)|$ is the length of the shortest vector in the lattice generated by $b_i(i)$, $b_{i+1}(i), \dots, b_n(i)$. Thus by Minkowski's theorem,

$$|T| \leq n^{n-i+1} (n-i+1)^{\frac{1}{2}(n-i+1)} \leq n^{\frac{3}{2}(n-i+1)}$$

Hence the lemma.

Theorem 1. The algorithm $CLP(n; b_0, b_1, \dots, b_n)$ with input integral vectors all of length at most $\sqrt{B}$ returns the closest point of $L(b_1, \dots, b_n)$ to b_0 in at most $O((4n)^{\frac{3}{2}n})$ arithmetic operations. Further all numbers produced by the algorithm are rationals with numerator and denominator expressible in $O(n^2(\log B + \log n))$ bits each.

Proof: It is easily seen from the algorithm that T(n) the number of arithmetic operations is bounded by

$$\begin{aligned} T(n) \leq\ & \text{number of arithmetic operations for SHORTEST } (n; \dots) \\ & + n^{\frac{3}{2}(n-i+1)}\ T(i-1) \\ & + q(n) \end{aligned}$$

where q(n) is a polynomial in n, the second term is the number of operations needed to solve the lower dimensional problems whose number is at most $n^{\frac{3}{2}(n-i+1)}$ by the last proposition.

Using Theorem 1, Section 2 we get the first part of the current theorem. The bound on the size of the numbers is obtained without much difficulty. The enumeration process, it can be easily seen does not involve integers that are too large. We omit this proof here.

Section 5. Integer Programming.

Integer programming again is the following problem:

(1) Given m×n and m×1 matrices A and b of integers, determine whether there is a x in $\mathbb{Z}^n$ such that $Ax \leq b$. Using a result of Lenstra (81), we can reduce it to the following problem:

(2) Given $b_1, b_2, \dots, b_n$ independent vectors in $\mathbb{Z}^n$, A m×n and b m×1 of integers determine whether there is an x in $L(b_1, b_2, \dots, b_n)$ such that $Ax \leq b$, where the following additional conditions are satisfied:

(3) a $p \in \mathbb{R}^n$, r and R reals such that
  (a) $R/r \leq 2n^{3/2}$
  (b) $B(p,r) \subseteq \{x \in \mathbb{R}^n, Ax \leq b\} \subseteq B(p,R)$.

(B(q,s) = ball of radius s with q as center).

We proceed as follows: Apply SHORTEST to $b_1, \dots, b_n$. Let now $|b_i(i)| = \max_j |b_j(j)|$. Then there is clearly a point b of $L(b_1, \dots, b_n)$ (by proposition 1 of section 4) such that $|b-p| \leq \frac{n}{2}|b_i(i)|$. We consider two cases: (as in Lenstra)

Case 1: $r \geq \frac{n}{2}|b_i(i)|$. Then clearly the answer to question 2 is Yes and we stop.

Case 2: $r < \frac{n}{2}|b_i(i)|$ whence $R < n^{5/2}|b_i(i)|$. In this case, we argue as in the last section that there are not too many values of $\lambda_i, \lambda_{i+1}, \dots, \lambda_n$ integers such that $\sum_{j=1}^{n} \lambda_j b_j$ belongs to B(p,R). We then enumerate all these values of $\lambda_i, \dots, \lambda_n$ and for each, solve a (i-1) dimensional problem.

To push through this recursion, we need to solve the following generalization of problem (2):

(4) Given $b_1, b_2, \dots, b_n$ independent vectors in $\mathbb{R}^n$, A,b m×n and m×1 matrices of integers and b_0 in $\mathbb{R}^n$, does there exist an x such that $Ax \leq b$ and $(x-b_0 \in L(b_1, b_2, \dots, b_n)$? We will call problem (4) generalized integer programming GILP for want of another name. We solve it by the following procedure:

procedure. $GILP(n; b_0, b_1, b_2, \dots, b_n; A, b)$.

Comment. See description of problem (4) above. The procedure returns Yes or No.

1. Use Lenstra's result which applies a suitable linear transformation on the space and ensures condition (3). We refer to the transformed quantities by their old names $b_0, b_1, b_2, \dots, b_n$; A,b.

2. $\{b_1, b_2, \dots, b_n\} \leftarrow$ SHORTEST $\{b_1, b_2, \dots, b_n\}$

```
3.  Let $b_i(i) = \max_{j=1}^{n} |b_j(j)|$.
4.  if $r \geq \frac{n}{2}|b_i(i)|$ then return Yes
    Comment. We may now assume that $r \leq \frac{n}{2}|b_i(i)|$
    and $R \leq n^{5/2}|b_i(i)|$.
5.  if i=1 then do.
    for each candidate $\lambda_1,\lambda_2,\ldots,\lambda_n$ integers do.
    Comment. Enumeration is explained later.
      if $b_0 + \sum_{j=1}^{n} \lambda_j b_j = x$ satisfies $Ax \leq b$ then return Yes.
                                                     end
        Return No
        end
    for each candidate $\{\lambda_i,\lambda_{i+1},\ldots,\lambda_n\} \in \mathbb{Z}^{n-i+1}$
    do: Comment. We explain later what the
    candidates are.
6.      $b_0 \leftarrow b_0 + \sum_{j=i}^{n} \lambda_j b_j$
7.      $\bar{b}_0 \leftarrow$ projection of $b_0$ onto span $\{b_1,b_2,\ldots,$
        $b_{i-1}\}$.
8.      $V \leftarrow \{v \in \mathbb{R}^n$: projection of v orthogonal to
        span $\{b_1,\ldots,b_{i-1}\} = b - \bar{b}_0\}$
Comment. Since we are "fixing" $\lambda_i,\lambda_{i+1},\ldots,\lambda_n$,
we are fixing the projection of our final potential candidate $(b_0 + \sum_{j=1}^{n} \lambda_j b_j)$ onto the orthogonal complement of span $\{b_1,b_2,\ldots,b_{i-1}\}$. Thus
we are only looking for candidates in V. V is
(i-1) dimensional affine set.
9.      Taking $b_1(1)/|b_1(1)|,\ldots,b_{i-1}(i-1)/|b_{i-1}(i-1)|$
        as the basis of span $\{b_1,b_2,\ldots,b_{i-1}\}$ express
        $\{x: Ax \leq b\} \cap V$ as $\{y: A'y \leq b'\}$ where A' is m' by
        (i-1). Let $\bar{b}_0$ be $b_0'$ in this coordinate system, $b_j$ be $b_j'$ for j=1,2,...,i-1.
10.  if $(GILP(i-1;b_0',b_1',b_2',\ldots,b_{i-1}',A',b')$
        returns Yes then return Yes.
        end
        Return No
        end GILP
```

As usual, we first explain the enumeration process. At the beginning of Step 5, we may assume that R is at most $|b_i(i)| \cdot n^{5/2}$. Thus the vectors that are of the form b_0+a, a in $L(b_1,\ldots,b_n)$ which could belong to the polytope $\{x: Ax \leq b\}$ all have the property that $|b_0+a-p| \leq |b_i(i)|n^{5/2}$. Hence they must each be of length at most $n^{5/2}|b_i(i)|$ in the direction of $b_n(n)$ which says that we need to try at most

$$3.n^{5/2} \frac{|b_i(i)|}{|b_n(n)|} \text{ values of } \lambda_n.$$

Arguing in a similar vein (cf Proposition 5 of section 2 and Proposition 2 of section 4), the number of candidates for $\lambda_i,\lambda_{i+1},\ldots,\lambda_n$ is at

most $\left| 2^{n-i+1} n^{5/2(n-i+1)} \prod_{j=i}^{n} |b_i(i)|/|b_j(j)| \right|$

which again by Minkowski's theorem is at most

$$((1 \cdot 29)n)^{3(n-i+1)}$$

That the algorithm works correctly is very easy to check.

Theorem 1: The algorithm GILP(n;...) on an input of length L, correctly solves the n-generalized integer programming problem in $O((1 \cdot 3n)^{3n})$ arithmetic operations. Each integer produced by the algorithm is $O(n^{5 \cdot 1\ n}.L)$ bits in size and thus the overall running time is $O(n^{9n}\ L \log(L))$.

Proof. The first part of the theorem follows by an argument very similar to the proof of Theorem 1 of section 4. The second part is a little trickier. By going through the construction to "round out" a polytope due to Lenstra (1981), one finds that this increases the number of bits by at most a factor of $n^2 \log n$. This is because Lenstra's algorithm obtains the affine transformation that rounds out the polytope $\{x: Ax \leq b\}$ by mapping (n+1) of its vertices (in n dimensions) to (0,0,0,...,0), (1,0,...0), (0,1,...,0) ..., (0,0,...,0,1). The claim above easily follows by reckoning a bound on the size of numbers needed to express the vertices and then observing that the matrix associated with the linear transformation is simply S^{-1} where S is a n×n matrix with each row being one of the (n+1) vertices. The algorithm SHORTEST then increases the sizes by at most a factor of $O(n^2 \log n)$ by Lemma 3 of section 3. But when the algorithm SHORTEST is finished, the sizes are much smaller by lemma 1 of section 3, and thus it is easily checked that the process of projecting to a lower dimensional problem does not increase the sizes beyond the bound obtained for SHORTEST. Thus each reduction in the dimension of the problems increases the size of numbers by at most a factor of $O(n^4(\log n)^2)$ which is $O(n^{4 \cdot 1})$. Since this happens at most n times, the sizes of the numbers are all bounded by $O(n^{(5 \cdot 1)n}L)$. The last sentence in Theorem 1 follows since the time taken to add, subtract, multiply or divide 2 N-bit integers is O(N log N) (Aho, Hopcroft and Ullmann (1974).

Remark: It is unfortunate that in this algorithm the bound we derive on the size of the integers is exponential in n. While this does not increase the order of the running time of the algorithm, it would be nice to prove a polynomial bound on the sizes of the integers involved, perhaps after modifying the algorithm slightly. The obvious try in this direction does not work - i.e., if the polytope $\{x: Ax \leqq b\}$ satisfies roundness conditions (3), it is trivially seen that its projections onto affine sets do not necessarily satisfy (3).

Section 6:

While we have improved substantially the theoretical bounds on the running times of known algorithms, the bounds we give may still seem too long in practice. However, all the algorithms given here are expected to run a lot faster than dictated by the theoretical bounds. Primarily, this is because the LLL algorithm used here to initialize the procedure SHORTEST yields in practice much shorter vectors than the bounds proved by them; in addition it should be the case that the procedure enumerate tries a lot fewer cases on the average than we can prove using Minkowski's Theorem. Of course all of these statements have to be tested empirically.

Section 7: Complexity Issues.

It was conjectured in Lenstra (1981) that the problem of finding a shortest vector in a lattice $L = L(b_1,\dots,b_n)$ given $b_1,\dots,b_n$ is NP-hard. This conjecture is still open. Van Emde Boas has proved (among others) the following language to be NP-complete:

$$L_2\text{-Closest} = \{(b_0,b_1,\dots,b_n;K) \mid b_0,b_1,\dots,b_n \in \mathbb{Z}^n \text{ and } \exists b \in L(b_1,\dots,b_n) \text{ such that } |b-b_0| \leqq K\}.$$

We do not solve the conjecture here, but relate the complexity of finding the shortest vector to L_2-closest. First define a language

$$L_2\text{-SHORTEST} = \{(b_1,\dots,b_n;K) \mid \exists b \in L(b_1,\dots,b_n),\ b \neq 0 \text{ such that } |b| \leqq K\}.$$

Theorem 1: Given $b_0,b_1,\dots,b_n$ in $\mathbb{R}^n$, with polynomially many calls to a subroutine for deciding membership in L_2-Shortest and polynomial additional time we can find a vector y in $L(b_1,\dots,b_n)$ such that for all y' in $L(b_1,\dots,b_n)$,

$$|y-b_0| \leqq n\cdot|y'-b_0|$$

Remark. The theorem asserts that the problem of finding an approximate closest vector to within a factor of n is polynomial-time Turing (Cook) reducible to L_2-Shortest. The reduction given is essentially Turing-it invokes more than one call to the subroutine. We show first that given a subroutine that accepts L_2-shortest, we can actually find a shortest vector in a lattice. Suppose $L=L(b_1,\dots,b_n)$, $b_i \in \mathbb{Z}^n$ independent is the lattice in which we want to find a shortest nonzero vector. Define

$$(1) \quad \ell = \left|(\sqrt{n}\,(d(L))^{\frac{1}{n}})^4\right|$$

Let τ be the linear transformation given by

$$\tau = \begin{pmatrix} \ell^n+\ell^{3n} & & & \bigcirc \\ & \ell^{n-1}+\ell^{3n} & & \\ & & \ddots & \\ \bigcirc & & & \ell^1+\ell^{3n} \end{pmatrix}$$

(τ multiplies the ith coordinate by $(\ell^{n+1-i}+\ell^{3n})$.

Lemma: Suppose $L=L(b_1,\dots,b_n)$ where $b_i \in \mathbb{Z}^n$ and are independent and define ℓ and τ as in (1) and (2). Then for $L^*=\tau L$, any shortest nonzero vector of L^* must be of the form

$$(3) \quad Y = (\ell^{3n}+\ell^n)y_1, (\ell^{3n}+\ell^{(n-1)})y_2, \dots, (\ell^{3n}+\ell)y_n)$$

where $(y_1,y_2,\dots,y_n)$ is a shortest and nonzero vector of L and for any other shortest vector $(y_1',y_2',\dots,y_n')$ of L, we have

$$(4) \quad |y_{i_0}| < |y_{i_0}'| \text{ for } i_0 = \min_{i=1}^{n} \{i : |y_i| \neq |y_i'|\}$$

(In other words $(|y_1|,\dots,|y_n|)$ is the lexicographically least among the shortest vectors of L).

Proof: Suppose $\lambda_1(L)$ and $\lambda_1(L^*)$ are the lengths respectively of the shortest nonzero vectors of L and L*. Then clearly,

$$(5) \quad \lambda_1(L^*) \leqq (\ell^{3n}+\ell^n)\,\lambda_1(L).$$

Suppose now Y is a shortest vector in L* and the corresponding $(y_1,y_2,\dots,y_n)$ (according to (3)) is not a shortest vector of L. Then

$$|Y| \geqq \ell^{3n}(|(y_1,y_2,\dots,y_n)|) \geqq \ell^{3n}(\lambda_1(L)+1)$$
$$= \ell^{3n}\lambda_1(L)+\ell^{3n} > \ell^{3n}\lambda_1(L)+\ell^n\lambda_1(L) \text{ (from (1))}.$$

This contradicts (5) and hence $y=(y_1,y_2,\dots,y_n)$ must be a shortest nonzero vector of L. Further, $|Y|^2=(\ell^{3n}+\ell^n)^2y_1^2+\dots+(\ell^{3n}+\ell)^2y_n^2$. Suppose $y'=(y_1',\dots,y_n')$ is another shortest vector of L and (4) is violated. Then $y_{i_0} \geqq y_{i_0}'+1$. It can be seen easily that this together with the fact that $|y_j'| \leqq \ell$ for all j (by the definition of ℓ in (1) and Minkowski) implies that $Y'=\tau y'$ is shorter than Y in L*-a contradiction. Thus $(|y_1|,\dots,|y_n|)$ must be lexicographically least among all the shortest vectors of L. From (3) and the fact that a shortest nonzero vector $y=(y_1,\dots,y_n)$ of L must satisfy $|y_j| \leqq \ell^{1/4}$ for all j, we see easily that if $|Y|^2$ is given, then $(|y_1|,|y_2|,\dots,|y_n|)$ can be determined: Expand the integer $|Y|^2$ to

the base ℓ to write
$|Y|^2 = \sum_{j=0}^{6n} \alpha_j \ell^j$; then $y_1^2 = \alpha_{2n}$, $y_2^2 = \alpha_{2(n-1)}, \ldots, y_n^2 = \alpha_2$
as is easily seen.

All of this has only given us $(|y_1|, |y_2|, \ldots, |y_n|)$ for some shortest vector of L, we still need the signs of the components. Towards this end, first note that L* has the property that if Y is a shortest vector of L*, then for any other shortest vector Y' of L*, $|Y_i| = |Y_i'|$ (by (4)). Let $(|Y_1|, |Y_2|, \ldots, |Y_n|)$ be the magnitudes of the coordinates of a shortest vector Y of L* obtained by the above procedure. Consider the (n-1) dimensional lattice:

$$L' = L^* \cap \{x: \; x_1|Y_2| - x_2|Y_1| = 0\}$$

Clearly, $\lambda_1(L') = \lambda_1(L^*)$ iff there is a shortest vector of L* with the first two coordinates positive. Let $L'' = L^* \cap \{x: \; x_1|Y_2| + x_2|Y_1| = 0\}$. Then $\lambda_1(L'') = \lambda_1(L^*)$ iff there is a shortest vector of L* with the first two coordinates of opposite signs. So, we do the following: using our subroutine for L_2-Shortest, we check if $\lambda_1(L') = \lambda_1(L^*)$. If so we find (recursively) a shortest vector in L' and hence figure out a shortest vector of L*, then of L. If not, we find (recursively) a shortest vector of L" and do likewise. Note that to solve the problem of finding a shortest vector in n-dimensions, we solve one instance of the corresponding (n-1) dimensional problem plus polynomially many calls to L_2-shortest.

Lemma 2: With polynomially many calls to a subroutine accepting the language L_2-Shortest and polynomial additional time, we can find a shortest nonzero vector in a lattice.

Proof. We can construct from the input lattice L, the lattice L* discussed above. For L*, we may find $(\lambda_1(L^*))^2$ $(=|Y|^2$ in the discussion above) by binary search using polynomially many calls to the language oracle. This then gives us a shortest vector of L as described above.

Remark: A lemma similar to the one above holds for most known NP-complete languages and several other ones-like linear programming. For example, it is easy to see by using self-reducibility that given an algorithm to test whether a given Boolean formula is satisfiable, we may use it to find a satisfying assignment. This speaks for the versality of the language SAT. (the set of satisfiable Boolean formulas). It is interesting that the language L_2-shortest not yet known to be NP-complete has this versatality.

We now study the relationship between the problem of finding a closest vector of a lattice in $\mathbb{R}^n$, to a given point in $\mathbb{R}^n$ (called the "in homogeneous problem") to that of finding a shortest nonzero vector of a lattice (called the "homogeneous problem"). The device we use to relate these two may be called the process of "homogenaisation". The technique has been used, for example, in polyhedral theory. The idea is to relate the inhomogeneous problem for a lattice L in n dimensions to a homogeneous problem for a lattice L' constructed from L in (n+1) dimensions.

Suppose we are given $b_1, b_2, \ldots, b_n, b_0$ in $\mathbb{Z}^n$ and are asked to find a point b of $L = L(b_1, \ldots, b_n)$ which is approximately (to be defined later) closest (in Euclidean distance) point of L to b_0. We first check whether b_0 is in L by using, for example the polynomial-time algorithm of Bachem and Kannan (1979) to solve linear diophantine equations. If so, we may stop. Otherwise we find (using the subroutines for the homogeneous problem) $\lambda_1(L)$ (the length of a shortest nonzero vector of L: Caution: this may be irrational, so we will only find an approximation to it in the actual algorithm, but to simplify the current discussion, assume we know $\lambda_1(L)$ exactly.). We then consider the lattice L' in Q^{n+1} generated by $b_i' = (b_i, 0)$ for $i = 1, 2, \ldots, n$ and $b_{n+1}' = (b_0, (\cdot 51)|\lambda_1(L)|)$. We find a shortest nonzero vector $v = (v_1, \ldots, v_{n+1})$ of L'. This gives us information about the vector closest to b_0 in L as summarized by the following lemma:

Lemma 3: Suppose $L = L(b_1, \ldots, b_n)$ is a lattice in $\mathbb{Z}^n$ and b_0 in $\mathbb{Z}^n$ is not in L. Let L' be as defined above and let $v = (v_1, \ldots, v_{n+1})$ be a shortest nonzero vector of L' with $v_{n+1} \leq 0$. Then if $v_{n+1} = 0$,

$$|b_0 - b| \geq \cdot 8 |\lambda_1(L)| \text{ for all b in L.}$$

If $v_{n+1} \neq 0$ then $v_{n+1} = -(\cdot 51)|\lambda_1(L)|$ and $(v_1, v_2, \ldots, v_n) + b_0$ is the closest vector in L to b_0.

Proof. The shortest vector $v = (v_1, v_2, \ldots, v_{n+1})$ must clearly satisfy $|v_{n+1}| \leq |\lambda_1(L)|$ because there is a vector of length $|\lambda_1(L)|$ in L and hence in L'. Thus $v_{n+1} = 0$ or $\pm(\cdot 51)|\lambda_1(L)|$. Without loss of generality, we assume $v_{n+1} \leq 0$ and hence is 0 or $-(\cdot 51)$ $|\lambda_1(L)|$. Let b be a closest point of L to b_0 and $b = \sum_{j=1}^{n} \alpha_j b_j$. If $|b - b_0| \leq \cdot 8 |\lambda_1(L)|$, then

$$\left|\sum_{j=1}^{n} \alpha_j b_j' - b_{n+1}'\right| \leq |\lambda_1(L)| \; ((\cdot 8)^2 + (\cdot 51)^2)^{\frac{1}{2}}$$
$$< |\lambda_1(L)|$$

and thus the shortest vector v of L' will have $v_{n+1} = -(\cdot 51)\frac{|\lambda_1(L)|}{2}$ and is nonzero. This proves the first statement.

For the second, clearly if $v_{n+1} \neq 0$, $v_{n+1} \leq 0$ implies $v_{n+1} = -\cdot 51 |\lambda_1(L)|$. Then v equals $(-b_{n+1}' + \sum_{j=1}^{n} \beta_j b_j')$ and the last component of v being fixed at absolute value $\cdot 51\ (|\lambda_1(L)|$, will be shortest when $\sum_1^n \beta_j b_j$ is closest to b_0. Thus the lemma is proved.

Acknowledgement: I wish to thank Gary Miller for helpful discussions.

References

A. Aho, J. Hopcroft and J. Ullmann, "The design and analysis of computer algorithms", (1974)

Van Emde Boas, "Another NP-complete problem and the complexity of computing short vectors in a lattice", IEEE Transactions on Information Theory, Vol. IT-28, No. 2 (1982).

J.W.S. Cassels, "An Introduction to the geometry of numbers", Springer, Heidelberg (1959).

Conway and Sloane, "Fast quantizing and decoding algorithms for lattice quantizers and codes", IEEE Transactions on Information Theory, Vol. IT-28, No. 2, (1982).

R. Kannan and A. Bachem, "Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix", SIAM J. on Comp. 8(1979).

J.C. Lagarias, "Computational complexity of simultaneous diophantine approximation problems", 23rd FOCS (1982).

A.K. Lenstra, H.W. Lenstra Jr. and L. Lovasz, "Factoring polynomials with rational coefficients", Report 82-05, Mathematisch Instituut, Universiteit Amsterdam (1982).

H.W. Lenstra, Jr., "Integer programming with a fixed number of variables", Report 81-03, Mathematisch Instituut, Universiteit ban Amsterdam (1981).

A. Shamir, "A polynomial-time algorithm for breaking the basic Merkle-Hellman crypto system", 23rd FOCS (1982).