Florida State University Libraries

Electronic Theses, Treatises and Dissertations

The Graduate School

2008

Factoring Univariate Polynomials over the Rationals

Andrew Novocin

Follow this and additional works at the FSU Digital Library. For more information, please contact lib-ir@fsu.edu

FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

FACTORING UNIVARIATE POLYNOMIALS OVER THE RATIONALS

By

ANDREW NOVOCIN

A Dissertation submitted to the
Department of Mathematics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Spring Semester, 2008

The members of the Committee approve the Dissertation of Andrew Novocin defended on April 9, 2008.

Mark van Hoeij
Professor Directing Dissertation

Robert van Engelen
Outside Committee Member

Amod Agashe
Committee Member

Ettore Aldrovandi
Committee Member

Paolo Aluffi
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

This thesis is dedicated to my parents, Norb and Marie,
on the occasion of their 50th year of being alive
and their 30th year of being married.

TABLE OF CONTENTS

Abstract v

1. Brief History of Factoring Polynomials over the Rationals 1
- 1.1 Main Algorithmic Problem: 1
- 1.2 Zassenhaus' Algorithm 1
- 1.3 The LLL Paper 4
- 1.4 The van Hoeij Algorithm 8
- 1.5 The Main Goal of This Thesis 13

2. Factoring with switch complexity independent of coefficient size 14
- 2.1 Partial Reductions 14
- 2.2 Partial Reduction of a Special Matrix 17
- 2.3 New Bounds for Factoring in $\mathbb{Q}[x]$ 20
- 2.4 Conclusion of Chapter 21

3. Factoring with switch complexity independent of degree and coefficient size 22
- 3.1 Why go on? 22
- 3.2 Introducing the Active Determinant 23
- 3.3 How to read this chapter 24
- 3.4 Initialize 29
- 3.5 Hensel Step 32
- 3.6 Check If Solved 35
- 3.7 Remove Vectors 38
- 3.8 Select Next Coefficient 41
- 3.9 pLLL the Probationary LLL run 45
- 3.10 Scale Up 54
- 3.11 LLL 59
- 3.12 Decide If Vector Added 60
- 3.13 Zassenhaus' Algorithm 67
- 3.14 Switch Complexity of this Algorithm 67

4. Why this thesis is interesting 68
- 4.1 Early Termination 68
- 4.2 Conclusion 69
- 4.3 Future Improvements 70

REFERENCES 71

BIOGRAPHICAL SKETCH 72

ABSTRACT

This thesis presents an algorithm for factoring polynomials over the rationals which follows the approach of the van Hoeij algorithm. The key theoretical novelty in our approach is that it is set up in a way that will make it possible to prove a new complexity result for this algorithm which was actually observed on prior algorithms. One difference of this algorithm from prior algorithms is the practical improvement which we call early termination. Our algorithm should outperform prior algorithms in many common classes of polynomials (including irreducibles).

CHAPTER 1

Brief History of Factoring Polynomials over the Rationals

Throughout this thesis we will explore algorithms for factoring univariate polynomials with rational coefficients. Given a polynomial in $\mathbb{Q}[x]$ we may express it in $\mathbb{Z}[x]$. Also if a polynomial over $\mathbb{Z}$ is not separable then a simple GCD based squarefree factorization can be performed.

1.1 Problem Statement:

Input: $f \in \mathbb{Z}[x]$, separable, and monic[1] of degree N.

Output: $g_1, \ldots, g_k$ irreducible in $\mathbb{Z}[x]$, with $f = g_1 \cdots g_k$.

In this chapter we will present three important algorithms used for solving this problem: Zassenhaus' algorithm, the LLL algorithm, and van Hoeij's algorithm.

Notation: Throughout the paper we use the terms 'theoretical' and 'complexity' when referring to a proven bound for the number of CPU operations an algorithm might need. This contrasts the term 'practical' which is used when talking about an algorithm's actual performance on examples. If there exists examples where a theoretical complexity bound matches an algorithm's actual performance then we call this bound sharp.

1.2 Zassenhaus' Algorithm

Zassenhaus' algorithm for factoring polynomials is the earliest of its kind and it has many nice features. Here is a summary of the important ideas:

1. If $g \in \mathbb{Z}[x]$ divides f then the coefficients of g are smaller than some M, a bound, that we can compute. (e.g. the Mignotte bound in [5] page 156)

2. If $g \in \mathbb{Z}[x]$ divides f then g can be reconstructed when $g \bmod p^a$ is known for some $p^a > 2M$.

3. Factor $f = f_1 \cdots f_r$ over $\mathbb{Z}_p$ (the p-adic integers). There are only finitely many monic factors of f in $\mathbb{Z}_p[x]$. Each is of the form $g_v := \prod f_i^{v_i}$ for some 0–1 vector $v = (v_1, \ldots, v_r)$.

[1] The monic assumption is for notational purposes, see the Remark at the end of section 1.2.3 for details

4. $f_1, \dots, f_r$ (and hence g_v) are not known exactly, but are only known mod p^a. That's enough using idea 2.

1.2.1 Sketch of the Algorithm

Definition 1. *If $g \in \mathbb{Z}[x]$ or $g \in \mathbb{Z}_p[x]$ then g mods p^a denotes the polynomial with coefficients in $(\frac{-p^a}{2}, \frac{p^a}{2}]$ that is congruent to g modulo p^a. The **s** in mods refers to **symmetric remainder**.*

Definition 2. *Let $f_1, \dots, f_r$ be the p-adic factors of f and $\tilde{f}_1, \dots, \tilde{f}_r$ their approximations of accuracy a (i.e. $f_i \equiv \tilde{f}_i$ mod p^a). Hensel Lifting is a procedure which increases the accuracy of the $\tilde{f}_i$, a.*

Let $f_1, \dots, f_r \in \mathbb{Z}_p[x]$ denote the p-adic factors.

Compute M = bound for coefficients of factors in $\mathbb{Z}[x]$.

Then compute the p-adic factors mod p^a for some $p^a > 2M$ (first compute the f_i mod p, and then mod p^a by Hensel lifting).

1. Given some 0–1 vector $v \in \{0,1\}^r$ one can decide if $g_v := \prod f_i^{v_i}$ is in $\mathbb{Z}[x]$ or not, by checking if g_v mods p^a divides f in $\mathbb{Z}[x]$.

2. A factor in $\mathbb{Z}[x]$ can be computed efficiently if its 0–1 vector v is known: Take the f_i with $v_i = 1$ and multiply them mods p^a.

1.2.2 Two Problems with Zassenhaus' Algorithm

Problem 1: Overshooting the Hensel Lifting. Let $g_1, \dots, g_k$ be the true factors of f in $\mathbb{Z}[x]$ and let $f_1, \dots, f_r$ be the local factors (over the p-adic integers). The Zassenhaus algorithm applies Hensel lifting to determine $\tilde{f}_1, \dots, \tilde{f}_r$ with a p-adic accuracy a that is guaranteed to be high enough to recover any potential factor of f in $\mathbb{Z}[x]$.

However, the problem is that this p-adic accuracy, a, is often much higher than what was actually necessary to recover all the factors $g_1, \dots, g_k$. This implies that Zassenhaus' algorithm often wastes CPU time on Hensel Lifting. In practice it frequently happens that f has one large factor, say g_1, and zero or more small factors, say $g_2, \dots, g_k$. Then, to recover $g_1, \dots, g_k$ we do not need p^a to be larger than twice the largest coefficient of g_1. All we need is that p^a is larger than twice the largest coefficient in $g_2, \dots, g_k$. This suffices to reconstruct $g_2, \dots, g_k \in \mathbb{Z}[x]$ from their modular images, after which the remaining factor g_1 can be determined by a division in $\mathbb{Z}[x]$.

It is easy to give examples where this latter a is ten times smaller than the a used in Zassenhaus' algorithm. Just multiply a small irreducible polynomial by a big one.

(Of course a needs to be large enough not only to find $g_2, \ldots, g_k$, but also large enough to prove that $g_1, \ldots, g_k$ are irreducible. More precisely, a needs to be large enough to solve the combinatorial problem mentioned in Problem 2 below. However, using [6] this can usually be done with much less Hensel lifting than what is used in Zassenhaus' algorithm.)

Problem 2: Exponential Search Time. When there are many local factors $f_1, \ldots, f_r$ it can take an exponentially long time to decide which of the local factors combine to form a rational factor g_i. The [6] algorithm presented a practical solution to this problem but made no attempt at a complexity estimate.

Practical Goal: We want to compute the factors $g_1, \ldots, g_k$ as quickly as possible, without Hensel lifting further than necessary. The easiest way to prevent lifting too far is to do these two steps after each Hensel lift:

1. Try to solve the combinatorial problem using [6]
2. and if this succeeds, try to reconstruct $g_2, \ldots, g_k$ from their modular images (and g_1 with a division).

Suppose that lifting to at least p^{100} was necessary to solve both steps 1 and 2. We use quadratic Hensel lifting, so a doubles each step. This means that we solve the problem once we lifted to p^{128}, which is close to optimal. So compared to Zassenhaus' algorithm we could save much CPU time on Hensel lifting (Hensel lifting often dominates the CPU time).

Key Practical Problem:

Couldn't this approach also be slower in some cases? After all: What about the time that was spent when step 1 or 2 failed when we lifted to p^{64}, or to p^{32}, etc.? Step 2 costs little CPU time, but if step 1 failed by not lifting far enough, couldn't we have wasted CPU time? We will resolve this practical problem in Chapter 3.

1.2.3 Complexity/CPU cost of Zassenhaus' algorithm

Notation: We use $\| f \|_\infty$ to represent the largest absolute value of coefficients of f.

If f is irreducible we end up trying 2^r (actually 2^{r-1} by skipping complements) combinations of $(v_1, \ldots, v_r) \in \{0,1\}^r$. Then the CPU time will be roughly: Cost(factoring f mod p) + Cost(Hensel lifting) + $2^r \cdot$tiny.

1. Cost(factoring mod p) depends polynomially on the degree N.
2. Cost(Hensel lifting) depends polynomially on N, $\log(\| f \|_\infty)$.

3. Using ideas from [1, 7], in particular section 3.1.1 in [1], testing one combination can usually be done in a tiny amount of time (even if N and $\log(\| f \|_\infty)$ are large).

Given some polynomial $f \in \mathbb{Z}[x]$ of degree N, the algorithm tries several primes p, and then chooses the one for which f has the fewest p-adic factors $f_1 \cdots f_r$. Usually $r << N$ and Zassenhaus' algorithm is fast, with Hensel lifting dominating the CPU time. But for polynomials that have large r at each p the algorithm suddenly takes exponential time.

The next algorithm for factoring polynomials is in the LLL paper which is presented in section 1.3.2. We will use its complexity for comparison with Zassenhaus' complexity.

Suppose f has degree $N \approx 200$, with ≈ 200 digit coefficients. For the best implementations of Zassenhaus' algorithm, as long as $r < 20$ then the precise value of r has little impact on the CPU time, it will take about a second either way. On the same example the LLL Algorithm For Factoring [8] would take a day. However, if we let $r = 64$ then [8] still only takes about a day but Zassenhaus would take an estimated 100,000 years (because of the exponential search).

Although the LLL algorithm for factoring is much better than Zassenhaus in this example, keep in mind that if we somehow knew which subset(s) of $f_1, \ldots, f_{64}$ to take, then Zassenhaus would only take 1 second which is much better than 1 day! Thus, the only thing that stands in the way of reducing CPU time from 1 day to 1 second are objects with *only 64 bits of data* (namely the $v \in \{0,1\}^r$ that encode the right subsets of $f_1, \ldots, f_r$).

The goal of van Hoeij's algorithm [6] is a quick way to compute this data. Before discussing the van Hoeij algorithm we will explore the two algorithms of note in the LLL paper [8].

Remark on non-monic case: The only changes that need to be made are as follows: Assume content$(f) = 1$ with $f \in \mathbb{Z}[x]$. We will find $f = l_c \cdot f_1 \cdots f_r$ for the p-adic factors, where l_c is the leading coefficient of f. If $v \in \{0,1\}^r$ corresponds to a rational factor compute it as primpart$(l_c \prod f_i^{(v)_i})$, where $(v)_i$ is the ith entry of v.

1.3 The LLL Paper

In [8] Lenstra, Lenstra and Lovász introduced a lattice reduction algorithm, which we shall refer to as the LLL algorithm. This algorithm is still one of the most used algorithms in computational algebraic number theory. The same paper also presented the first polynomial time algorithm for factoring polynomials, which came as a surprise at the time. Their factoring algorithm avoids the above mentioned combinatorial problem by constructing factors of f using the LLL algorithm.

1.3.1 Overview of Lattice Reduction

The purpose of this subsection is to introduce the LLL algorithm for lattice reduction (which we will call LLL) and list some notations and known facts (from [8]) which will be used throughout the thesis.

A lattice L is a discrete subset of $\mathbb{R}^m$ that is also a $\mathbb{Z}$-module. Let $b_1, \ldots, b_r \in L$ be a basis of L and denote $b_1^*, \ldots, b_r^* \in \mathbb{R}^m$ as the Gram-Schmidt orthogonalization over $\mathbb{R}$ of $b_1, \ldots, b_r$. Let $l_i = \log_{4/3}(\| b_i^* \|^2)$, and denote $\mu_{i,j} = \frac{b_i \cdot b_j^*}{b_j^* \cdot b_j^*}$. Note that $b_i, b_i^*, l_i, \mu_{i,j}$ will change throughout the algorithm sketched below.

Definition 3. *$b_1, \ldots, b_r$ is* LLL-reduced *if $\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2$ for $1 \leq i < r$. (The definition in [8] is slightly stronger, for convenience we only listed what is needed for this thesis. See also Remark 2 at the end of this section.)*

Algorithm 1 (Rough sketch of Lattice Reduction Algorithms)**.**

Input: *A basis $b_1, \ldots, b_r$ of a lattice L.*

Output: *An LLL-reduced basis of L.*

1. (Gram-Schmidt over $\mathbb{Z}$). *By subtracting suitable $\mathbb{Z}$-linear combinations of $b_1, \ldots, b_{i-1}$ from b_i make sure that $|\mu_{i,j}| \leq 1/2$ for all $j < i$.*

2. (LLL Switch). *If there is a k such that interchanging b_{k-1} and b_k will decrease l_{k-1} by at least 1 then do so.*

3. (Repeat). *If there was no such k in Step 2, then the algorithm stops. Otherwise go back to Step 1.*

That the above algorithm terminates, and that the output is LLL-reduced was shown in [8]. Step 1 has no effect on the l_i. In step 2 the only l_i that change are l_{k-1} and l_k. To illustrate step 2 in more detail, suppose that $c_1, \ldots, c_r$ is a basis of L obtained from $b_1, \ldots, b_r$ by applying step 2. So $c_k = b_{k-1}$ and $c_{k-1} = b_k$, and $c_j = b_j$ for the remaining j's.

Since $b_1, \ldots, b_r$ and $c_1, \ldots, c_r$ are bases of the same L, they have the same determinant (the product of $\| b_i^* \|$ for $i = 1, \ldots, r$) and hence

$$\| c_{k-1}^* \|^2 \| c_k^* \|^2 = \| b_{k-1}^* \|^2 \| b_k^* \|^2. \tag{1.1}$$

Step 2 is only taken if it decreases l_{k-1} by at least 1, so $\| c_{k-1}^* \|^2 \leq \frac{3}{4} \| b_{k-1} \|^2$. The vector b_k^* is obtained from b_k by reducing it modulo $\mathbb{R}b_1 + \ldots + \mathbb{R}b_{k-1}$ while c_{k-1}^* is obtained from b_k by reducing it modulo $\mathbb{R}b_1 + \ldots + \mathbb{R}b_{k-2}$. Hence b_k^* can not be longer than c_{k-1}^*. Combining these we find

$$\| b_k^* \|^2 \leq \| c_{k-1}^* \|^2 \leq \frac{3}{4} \| b_{k-1}^* \|^2 \tag{1.2}$$

which by equation (1.1) is equivalent to

$$\parallel b_{k-1}^* \parallel^2 \geq \parallel c_k^* \parallel^2 \geq \frac{4}{3} \parallel b_k^* \parallel^2.$$

The equations imply:

Observation 1. *An LLL switch can not increase* $max(l_1, \ldots, l_r)$*, nor can it decrease* $min(l_1, \ldots, l_r)$*.*

In summary, an LLL switch reduces l_{k-1} by at least 1, and increases l_k by the same amount (because of equation (1.1)). This way each LLL switch moves G-S (Gram-Schmidt) length towards the later vectors, while the sum of the logarithmic G-S lengths $l_1 + \cdots + l_r$ stays the same. Moreover, during the Lattice Reduction Algorithm, the highest G-S length can not increase, and the lowest G-S length can not decrease.

Consider the sum $\sum_i (r-i) l_i$. Each LLL switch reduces this sum by at least 1. Hence, if l^{in} was the value of this sum at the beginning of the computation, and l^{out} was the value at the end, then there can not have been more than $l^{\text{in}} - l^{\text{out}}$ LLL switches during the computation. The same idea was used in the proof of the complexity result given in [8, Proposition (1.26)]. Another useful fact is the following:

Fact 1. *Let* B *be a positive number. If* $b_1, \ldots, b_r$ *is a basis of* L *with* $\parallel b_r^* \parallel^2 > B$ *then any vector,* $v \in L$ *with* $\parallel v \parallel^2 \leq B$ *can be written as a* $\mathbb{Z}$*-linear combination of* $b_1, \ldots, b_{r-1}$*.*

In other words, $b_1, \ldots, b_r$ is a basis of some lattice L, and if the last vector has sufficiently large G-S length, then, in applications (including ours) where one is only interested in elements of L of squared length $\leq B$, one can remove the last basis element.

Fact 1 follows from the proof of (1.11) in [8], and is true regardless of whether $b_1, \ldots, b_r$ is LLL-reduced or not. However, if one chooses an arbitrary basis $b_1, \ldots, b_r$ of some lattice L, then it is unlikely that the last vector has large G-S length (after all, $\parallel b_r^* \parallel$ is the length of b_r reduced modulo all $\mathbb{R}$-linear combinations of $b_1, \ldots, b_{r-1}$). The effect of LLL reduction (Algorithm 1) is to move G-S length towards later vectors. So LLL reduction is very useful because if enough of this G-S length arrives at the last vector, then it can be discarded, which brings us one step closer to our target. (Our target is to solve the combinatorial problem of selecting the right $v \in \{0,1\}^r$ from 1.2.3.)

Remarks:

1. There are a number of lattice reduction algorithms that are variations of the LLL algorithm sketched above. We would like to present our complexity result in a way that is independent of which variation is used.

Each of these variations uses (at least asymptotically) the same number of LLL switches. The differences in complexity come from differences in the cost per LLL switch. So we will express our complexity result in terms of the number of LLL switches. This way our result will be compatible with each variation on the LLL algorithm.

2. Schönhage [10] gives a slightly different definition of reduced, called semi-reduced. This allows him to apply a divide-and-conquer strategy called block-wise reduction that reduces the asymptotic cost per LLL switch. With minor modifications, the results in this thesis carry through if one replaces 'reduced' by Schönhage's 'semi-reduced'.

3. There is also a new 'floating point LLL' algorithm by Phong Nguyen and Damien Stehlé [9], which is fast both in practice and in theory.

An LLL reduced basis has a nice property implied in Definition 3: Since $\| b_1^* \| = \| b_1 \|$ by the G-S process, we know that $\| b_1 \| \leq 2^{r-1} \| b_k^* \|$ for every possible k. But every nonzero vector, $v \in L$ is at least as big as the smallest G-S length, i.e. $\| v \| \geq \min(\| b_i^* \|)$, see [8]. These two facts combine to give us the property that every nonzero vector in L has length $\geq \frac{1}{2^{(r-1)/2}} \cdot \| b_1 \|$

1.3.2 A version of the factoring algorithm from LLL [8]

There are algorithms for approximating roots of a polynomial f. So let $\alpha = a + bi$ be an approximate[2] root of f found by such an algorithm. Then let C be some large (compared to N, $|\alpha|$, and the coefficients of f) real number. If we knew[3] the degree of the minpoly of α to be n and let L be the lattice spanned by the rows of this matrix:

$$M = \begin{bmatrix} 1 & 0 & \cdots & 0 & C \cdot \mathrm{Re}(\alpha^0) & C \cdot \mathrm{Im}(\alpha^0) \\ 0 & \ddots & & \vdots & \vdots & \vdots \\ \vdots & & \ddots & 0 & & \\ 0 & \cdots & 0 & 1 & C \cdot \mathrm{Re}(\alpha^n) & C \cdot \mathrm{Im}(\alpha^n) \end{bmatrix}$$

then we would know that there is a vector $v \in L$ with first $n+1$ entries being the coefficients of the minpoly of α and the final two entries being very small (not quite 0 since we only have an approximation). For a sufficiently large C and sufficiently accurate approximation of α, the LLL algorithm will discover this vector (it would be $\pm b_1$ in the output) and hence an irreducible factor of f.

Terminology: When we run LLL on a matrix like M we call this *feeding data* to LLL. (Here we have 'fed' the real and imaginary parts of $\alpha^0, \ldots, \alpha^n$ to LLL.)

[2] In [8] they use p-adic approximations rather than floating point approximations.
[3] We could start guessing that $n = 1$ and let n increase by one at a time. Although there is a better way, e.g. [10]

Let's revisit the example from the previous section: suppose f has degree $N \approx 200$, with ≈ 200 digit coefficients, and say $r = 64$ p-adic factors $f = f_1 \cdots f_{64}$. To construct an irreducible factor $g \in \mathbb{Z}[x]$ (worst case: $g = f$ if f is irreducible) with [8] means finding v, the corresponding vector, with lattice reduction. This vector could contain as much as $200 \cdot \log_2 10^{200} \approx 132,000$ bits of data, and LLL could take a day.

However, if we had $r = 64$ bits of data, $v = (v_1, \ldots, v_r) \in \{0, 1\}^r$ then we could compute the corresponding factor $g = \prod f_i^{v_i}$ in 1 second with Zassenhaus.

Main idea in [6]: Use LLL to compute $(v_1, \ldots, v_r)$ in a way that avoids computing any bits of information about the coefficients of g.

1.4 The van Hoeij Algorithm

Let $f = f_1 \cdots f_r \in \mathbb{Z}_p[x]$. The map

$$v \mapsto g_v = \prod f_i^{v_i}$$

that sends a 0–1 vector $v = (v_1, \ldots, v_r)$ to the corresponding factor of f turns additions into multiplications. For lattice reduction we need something that is linear, so we have to turn multiplications back into additions. One way to do that is using the following map:

$$g \mapsto \mathrm{Tr}_1(g)$$

where $\mathrm{Tr}_1(g)$ is the sum of the roots (with multiplicity) of g. So we get an additive map

$$\phi : v \mapsto \mathrm{Tr}_1(g_v) = \sum v_i \mathrm{Tr}_1(f_i)$$

from $\mathbb{Z}^r$ to the p-adic integers $\mathbb{Z}_p$. So lets take $t_i := \mathrm{Tr}_1(f_i) \in \mathbb{Z}_p$ for $i = 1, \ldots, r$ and look at this map

$$\phi : v = (v_1, \ldots, v_r) \mapsto \mathrm{Tr}_1(g_v) = v_1 t_1 + \cdots + v_r t_r$$

from $\mathbb{Z}^r$ to $\mathbb{Z}_p$. If $g_v \in \mathbb{Z}[x]$ then $\mathrm{Tr}_1(g_v)$ is an integer bounded by some b (assume for now that f is monic. For b we can then take N times a bound for the absolute values of the complex roots of f). Set

$$\tilde{t}_i := (t_i \text{ mods } p^a) \in \mathbb{Z}$$

Then

$$\mathrm{Tr}_1(g_v) = v_1 \tilde{t}_1 + \cdots + v_r \tilde{t}_r \ + \text{ small multiple of } p^a$$

for any of our target v's (the v's for which $g_v \in \mathbb{Z}[x]$).

Now $\mathrm{Tr}_1(g_v)$ is a coefficient of the factor g_v, but for efficiency we want to compute $(v_1, \ldots, v_r)$ without computing any coefficients of factors of f (recall the earlier discussion about 132,000 bits versus 64 bits in 1.2.3). So we take

$$s_i := \frac{\tilde{t}_i}{b} \in \mathbb{Q}$$

(the implementation rounds this to an integer for efficiency, but we'll skip that for simplicity).

Now let L^{in} be the lattice generated by:

$$(1, 0, \ldots, 0, s_1), \quad (0, 1, \ldots, 0, s_2), \; \ldots \; (0, 0, \ldots, 1, s_r)$$

and $(0, 0, \ldots, 0, \frac{p^a}{b})$.

Any target $v = (v_1, \ldots, v_r)$ corresponds to a vector

$$v' = (v_1, \ldots, v_r, \; \mathrm{Tr}_1(g_v)/b) \in L^{\mathrm{in}}.$$

All entries of v' are bounded by 1, so

$$\| \, v' \, \| \leqslant B := \sqrt{r+1} \qquad (B \text{ is a bit higher if we rounded})$$

(Note: in the later chapters the notation will be slightly different. There we will be dealing with squared G-S lengths and B will be $r+1$.) Let L_B be the span of all vectors in L^{in} of length $\leqslant B$, and we let π be the projection on the first r coordinates, then all of our target v's are in $\pi(L_B)$. Let $b_1, \ldots, b_{r+1}$ be an LLL reduced basis. Let $s := r+1$. As long as $\| \, b_s^* \, \| > B$ replace s by $s-1$. After these removals define $L := \mathrm{SPAN}_{\mathbb{Z}}\{b_1, \ldots, b_s\}$. If all vectors in $L^{\mathrm{in}} \backslash L_B$ are sufficiently large, then $L = L_B$. But we make no effort to make sure this is true, so instead we get a lattice L such that:

$$L_B \subseteq L.$$

Notation: Denote W as the span of our target v's. The 0–1 vectors corresponding to the irreducible factors of f in $\mathbb{Q}[x]$ form a basis of W. (The reduced echelon basis of W.)

Remark: Solving combinatorial problem $\Longleftrightarrow$ computing W.

For any vector, v, we let $\pi(v)$ denote the projection of v onto the first r entries. For any lattice $S = \mathrm{SPAN}_{\mathbb{Z}}\{v_1, \ldots, v_s\}$ we let $\pi(S)$ denote $\mathrm{SPAN}_{\mathbb{Z}}\{\pi(v_1), \ldots, \pi(v_s)\}$. We now know that:

$$W \subseteq \pi(L_B) \subseteq \pi(L).$$

W is the lattice we want, and L is the lattice we can get from LLL. Given L we can quickly test whether $\pi(L)$ equals W or not:

1. Check if the reduced echelon basis of $\pi(L)$ consists of 0–1 vectors, $v_1, \ldots, v_s$.
2. If so reconstruct $g_j = \prod f_i^{(v_j)_i}$ mods p^a and perform a trial division.
3. If each g_j divides f in $\mathbb{Z}[x]$ then $\pi(L) = W$.

If $\pi(L)$ equals W then we are done, and the resulting factors are irreducible regardless how many p-adic digits were used.

Prior factoring algorithms need some lower bound on the p-adic precision in order to prove that the factors are irreducible. The van Hoeij algorithm does not need such a bound, because of the following:

- The algorithm only terminates if it finds $\dim(\pi(L))$ factors in $\mathbb{Z}[x]$, whose product equals f.
- Any set of $\geqslant \dim(W)$ factors in $\mathbb{Z}[x]$, with product f, are automatically irreducible.
- $\pi(L) \supseteq W$ is true for any p-adic precision.

  (if we didn't use any digits at all we'd get $L = \pi(L) = \mathbb{Z}^r$. Using more digits brings $\pi(L)$ closer to W, but $\pi(L) \supseteq W$ will always hold, and termination only happens when $\pi(L) = W$).

Since no bounds on the p-adic accuracy are needed to prove that the output is irreducible, we can be very flexible with how many p-adic digits to use. However, we only find the factors when $\pi(L) = W$, so in order for the algorithm to terminate, we do need that $\pi(L)$ eventually becomes W.

So what if $\pi(L) \neq W$? We can gradually add more and more p-adic digits, but that may not be enough. Additional data may be needed. For instance, instead of Tr_1 (= sum of roots) we can also use Tr_2 (= sum of squares of roots), Tr_3 (= sum of cubes) etc.

One can prove that $\pi(L)$ will eventually become W if we keep using more and more "traces" Tr_i and p-adic digits, see [6].

1.4.1 Cost of van Hoeij's algorithm

In the paper [6] van Hoeij only proves the termination of his algorithm and no attempt to bound the complexity is even made. In practice though the algorithm has the same costs as Zassenhaus' algorithm except we replace the 2^r term by the cost of using LLL to solve the combinatorial problem. This is intuitively a problem related to the size of r, the number of local factors of f.

The cost of LLL dominates the algorithm's CPU cost if and only if r is large. It also dominates the theoretical complexity. So we introduce a term to help express this cost:

Definition 4. *We will call the number of LLL switches which take place throughout an algorithm the **Switch Complexity** of the algorithm.*

For example we can estimate the Switch Complexity of the Schönhage algorithm by $\mathcal{O}(N^2(N+\log(H)))$, where $H = \| f \|_\infty$.

1.4.2 Variations of the van Hoeij Algorithm

The van Hoeij algorithm is very flexible with what it accepts as data for solving the combinatorial problem, so long as eventually we have a basis of W. This raises the question of how many p-adic digits, and of how many traces, need to be fed to LLL in order to successfully solve the combinatorial problem. If one uses too few traces or p-adic digits then one may make only partial progress instead of fully solving the combinatorial problem. If one uses too many, one can end up solving the combinatorial problem using much more CPU time than would have been necessary. From a practical point of view, the latter (using too many traces/digits) is worse because the wasted time can not be recovered, while the former (using too few traces/digits) can be remedied by gradually adding more traces and/or digits.

In fact, even if one knew the exact number of p-adic digits and traces that need to be used in order to solve the combinatorial problem with one call to LLL, then as explained in [6, section 2.3 item 3] it could still be faster to use fewer traces/digits for the first call to LLL despite the resulting increase in the number of calls to LLL. Thus, [6, section 2.3 item 2] proposed to add only few traces at a time, while [6, section 2.3 item 3] suggested to use $\mathcal{O}(r^2)$ bits of data in the first call to LLL (a p-adic digit counts as $\log_2(p)$ bits).

The trade-off is that adding many traces/digits at a time will reduce the number of calls to LLL while increasing the cost of each call. Adding few traces/digits at a time reduces the cost of each LLL call, but the number of calls goes up. There are two extreme positions to make concerning this trade-off.

A. Minimize the number of calls to LLL. (In the version described in Theorem 4.3 in [3] this number is brought down to 1.)

B. Minimize the complexity of each individual call. (In the version described by Belabas [2], the cost of each LLL call is bounded by a polynomial that depends only on r, that is, a polynomial that is independent of both N and the coefficient size of f.)

The strategy proposed by Belabas in [2] takes side [B] to the extreme. It uses just one trace at a time, and adds only $\mathcal{O}(r)$ bits of data at a time. While this leads to a good complexity bound for each LLL call, one that depends solely on r, it becomes difficult to bound the number of LLL calls if one takes side [B]. We expect the number of LLL calls in Belabas' implementation to be $\mathcal{O}(r)$ for typical examples, however, it should be possible to construct examples where this number is significantly higher.

What was new about Belabas' strategy is the order in which the p-adic digits are used; this is done in such a way that each call to LLL will maximally benefit from the preceding LLL calls. This way the fact that

the number of LLL calls may be large does not hurt the practical performance of the algorithm. Indeed, the number of LLL calls appears to have little effect on the running times of Belabas' implementation; section 2.5.1 in [2] mentions that reducing the value of the parameter BitsPerFactor by a factor 2 (which should double the number of LLL calls) has only a minor impact on the computation timings.

However, having an unknown (and potentially large) number of LLL calls certainly complicates the problem of bounding the complexity. Thus, to get a complexity bound, the paper [3] took side [A]. In Theorem 4.3 in [3] it was shown that the combinatorial problem would be solved with a single LLL call if one applies LLL to a certain lattice (called the all-coefficients lattice in [3]). However, this lattice is as big as the one used in [8] and thus one ends up with the same complexity. The paper [3] also showed (Theorem 4.6 in [3]) how to bound the complexity for [B] (i.e. for Belabas' version, called "one coefficient at a time" in [3]). Although this bound was not spelled out explicitly (Theorem 4.6 in [3] only says "polynomially bounded"), if one follows the steps of the proof one sees that the complexity result for [B] in [3] is much worse than the bound for [A], which is ironic, because [B] runs very much faster than [A]. This is one of the key problems to be resolved in this thesis.

Key Theoretical Problem

Ever since the LLL paper in 1982, the algorithms with the best proved complexity bounds are not the algorithms which are the fastest in practice.

Notation: In [3] we see that for large enough a solving the combinatorial problem can be accomplished by finding an LLL-reduction of the rows of the following matrix, called the **All-Coefficients Matrix**:

$$\begin{pmatrix} & & & & & p^a \\ & & & & \iddots & \\ & & & p^a & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}.$$

Here the $*$ represent coefficients of the logarithmic derivative (multiplied by f) for each of the local factors.

The important result of [3] that we need is that they express an upper bound for the amount of Hensel Lifting needed before an LLL reduction of the rows of the All-Coefficients matrix ensures a solution of the combinatorial problem.

Remark: If a proved complexity bound for algorithm A is better than the proved bound for algorithm B it does *not* imply that there exists examples for which algorithm A is faster. It could be that algorithm B is faster but the complexity bound for B is not sharp.

1.5 The Main Goal of This Thesis

We would like to resolve the key theoretical problem by closing the gap between theory and practice. In the algorithm with best proved complexity, [10], and in the algorithm with best practical running times, [2, 6], the dominant term is the Switch Complexity. So we will only focus on reducing the switch complexity.

The goal is to show an algorithm for factoring polynomials with a Switch Complexity of $\mathcal{O}(r^3)$, which is asymptotically sharp. In chapter 2 we will introduce a method (inspired by Belabas' [B] strategy) for finding a reduction of the rows of a particular style of matrix with switch complexity independent of the size of the entries in the matrix. This method works for factoring polynomials when applied to the All-Coefficients matrix from [3] and has switch complexity $\mathcal{O}(Nr^2)$. Chapter 2 is taken largely from a joint preprint with Mark van Hoeij.

In Chapter 3 we expand on this method in an algorithm for factoring polynomials which has switch complexity $\mathcal{O}(r^3)$. This algorithm is presented in full detail and with complete and detailed proofs.

The same algorithm from Chapter 3 introduces a new practical feature we call Early Termination. This new feature allows us to always minimize the amount of Hensel Lifting without hurting the overall switch-complexity. In cases where the combinatorial problem can be solved before the factors themselves can be reconstructed (for instance many irreducible polynomials) we might be able to save a great deal of time, although in worst-case polynomials we do not see a benefit. This new algorithm should resolve our key practical problem and thus resolve both problems with Zassenhaus' algorithm listed in section 1.2.2.

CHAPTER 2

Factoring with switch complexity independent of coefficient size

2.1 Partial Reductions

This section presents a type of partial lattice reduction that we will later apply to factoring polynomials.

Definition 5. *Given a lattice $L \subseteq \mathbb{R}^m$ and a positive real number B, we call $S = b_1, \ldots, b_k \in L$ a B-reduced sequence for L if:*

1. $\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2$ *for* $1 \leq i < k$ *(Same as definition 3).*

2. $\| b_k^* \|^2 \leq B$.

3. *Every* $v \in L$ *with* $\| v \|^2 \leq B$ *is in* $SPAN_{\mathbb{Z}}(S)$.

Algorithm 2 (B-Reduce)**.**

Input: $V = b_1, \ldots, b_r$ *a basis of a lattice* L.

Output: *A* B*-reduced sequence for* L.

Algorithm: *The same as algorithm 1 in Section 1.3.1, except that whenever the last vector has squared G-S length* $> B$*, at any point during the computation, it is removed.*

Correctness of the algorithm is based on Fact 1 in subsection 1.3.1. Now consider the following problem. Let $\pi : \mathbb{R}^{m+1} \rightarrow \mathbb{R}^m$ be projection onto the first m coordinates, and $S = b_1, \ldots, b_r \in \mathbb{R}^{m+1}$ with $\pi(b_1), \ldots, \pi(b_r)$ already B-reduced. We would like to B-reduce S in a way that takes advantage of the fact that the first m entries are already B-reduced.

We could just apply algorithm 2 to S, but then the worst-case complexity would be the same as LLL. Specifically, if we applied algorithm 2 to S, then the *switch-complexity* may depend on the size of the last entries of the vectors of S. Recall that the switch-complexity is defined as the number of LLL-switches (step 2 of algorithm 1) that occur during the algorithm.

We will give a variant of algorithm 2 with a better switch-complexity, one that (under a mild assumption, see Theorem 1) only depends on B and r and not the size of the last entries. (This does not imply that the

overall complexity itself is independent of the size of last entries because we still work with numbers of that size. But a lower switch-complexity does imply a lower overall complexity because both algorithms have to work with vectors of similar size.) The algorithm uses an idea obtained from strategy B in [2].

Notation: Let $(v)_i$ denote the ith entry of any vector v.

Algorithm 3 (Gradual B-Reduce)**.**

Input: $S = b_1, \ldots, b_r \in \mathbb{R}^{m+1}$ *with* $\pi(b_1), \ldots, \pi(b_r)$ *already* B*-reduced.*

Output: *A* B*-reduced sequence.*

Algorithm:

1. *Let* d *be the smallest nonnegative integer for which* $|\frac{(b_i)_{m+1}}{2^{rd}}| \le 2^r$ *for* $i = 1, \ldots, r$*, where* $(b_i)_{m+1}$ *is the last entry of* b_i*.*

2. *Scale down the last entry of each vector* $(b_i)_{m+1} := \frac{(b_i)_{m+1}}{2^{rd}}$ *(for* $i = 1, \ldots, r$*) and set* $s := r$*.*

3. *Repeat the following* d *times:*

   (a) *If* $1 < max(|(b_1)_{m+1}|, \ldots, |(b_s)_{m+1}|)$ *(where* s *is the number of remaining vectors) then run algorithm 2 and let* s *be the number of remaining vectors.*

   (b) *(Gradually scale back up). Let* $(b_i)_{m+1} := 2^r\,(b_i)_{m+1}$ *(for* $i = 1, \ldots, s$*).*

4. *Run algorithm 2 and stop.*

Example for Algorithm 3:

Let $B := 10$.

Input: $S = b_1, b_2, b_3 := (0, 0, 1000), (1, 0, 333), (0, 1, 665)$. So $r = 3$, $m = 2$, and $\pi(S) = (0, 0), (1, 0), (0, 1)$ is B-reduced.

1. Find $d = 3$ so that $|\frac{1000}{2^{rd}}| \le 2^r$.

2. Initial Scale Down: $s := 3$, $2^{rd} = 512$ so $b_1 := (0, 0, \frac{1000}{512})$, $b_2 = (1, 0, \frac{333}{512})$, $b_3 = (0, 1, \frac{665}{512})$.

3a First Time. $1000/512 > 1$ so run algorithm 2 (basically just LLL) and now have $b_1 := (1, 0, \frac{333}{512})$, $b_2 = (0, 1, \frac{-335}{512})$, $b_3 = (-1, 1, \frac{83}{128})$ and $s = 3$.

3b. Scale the last entries by 2^r: $b_1 := (1, 0, \frac{333}{64})$, $b_2 = (0, 1, \frac{-335}{64})$, $b_3 = (-1, 1, \frac{83}{16})$

3a Second Time. $83/16 > 1$ so run algorithm 2 and now have $b_1 := (1, 1, \frac{-1}{32})$, $b_2 = (-2, 1, \frac{-1}{64})$, and $s = 2$ (the third vector was removed because LLL reduction pushed its squared G-S length over B).

3b. Scale the last entries by 2^r: $b_1 := (1, 1, \frac{-1}{4})$, $b_2 = (-2, 1, \frac{-1}{8})$.

3a Third Time. Now when running algorithm 2: b_1, b_2 are already B-reduced so: $s := 2$ and $b_1 := (1, 1, \frac{-1}{4})$, $b_2 = (-2, 1, \frac{-1}{8})$.

3b. Scale the last entries by 2^r: $b_1 := (1, 1, -2)$, $b_2 = (-2, 1, -1)$.

4. Again already B-reduced. So this call to algorithm 2 does nothing.

Output: b_1, b_2.

In order to see why this algorithm returns a B-reduced basis we need the following:

Lemma 1. *Let $\sigma : \mathbb{R}^{m+1} \to \mathbb{R}^{m+1}$ scale up the last entry by some factor $\delta > 1$. Let $S = b_1, \ldots, b_r$ and $\sigma(S)$ the image of S. Then $\| b_i^* \| \leq \| \sigma(b_i)^* \|$.*

The Lemma implies that removing a vector (during the call to algorithm 2 in step 3a) before the last entry was scaled back up to original scaling (by repeated calls to step 3b) will not cause removal of a vector whose original squared G-S length (as it was before step 2) had been $\leq B$.

Proof. Let $V_i = \{b_i - (a_{i-1}b_{i-1} + \cdots + a_1 b_1) \mid a_1, \ldots, a_{i-1} \in \mathbb{R}\}$, then b_i^* is just the shortest vector in V_i. Now the claim is that the shortest vector in V_i is not longer than the shortest vector in $\sigma(V_i)$. So let w be the shortest vector in $\sigma(V_i)$. There is some $v \in V_i$ with $\sigma(v) = w$. Let $w = (c_1, \ldots, c_m)$, then $v = (c_1, \ldots, c_m/\delta)$. Now $\| b_i^* \| \leq \| v \| \leq \| w \| = \| \sigma(b_i)^* \|$. □

Lemma 2. *In algorithm 3 every vector has squared G-S length $\leq 2^{3r}B$ at any time during steps 3 and 4. In particular, this holds for every removed vector at the time of its removal.*

Proof. Since running B-reduce cannot increase G-S lengths (by Observation 1 in Section 1.3.1) we need to decide how large a vector can possibly be just after scaling up by 2^r. But just before the scaling we know that $b_1, \ldots, b_s$ is B-reduced which implies that $\| b_1^* \|^2 \leq 2\| b_2^* \|^2 \leq \cdots \leq 2^{s-1}\| b_s^* \|^2 \leq 2^{s-1}B$, since the last vector is no larger than B and by the definition of an LLL-reduced basis (in Section 1.3.1). So just after scaling the largest G-S length can have squared length no greater than $(2^{2r})(2^{s-1}B) \leq 2^{3r}B$. □

Definition 6 (Progress)**.** *Suppose $b_1, \ldots, b_s$ is the current set of vectors at some point in the algorithm. We will define*

$$P(b_1, \ldots, b_s) = 0 \cdot l_1 + 1 \cdot l_2 + \cdots + (s-1) \cdot l_s + r(r-s)\log_{4/3}(2^{3r}B)$$

which we call the Progress *so far in the algorithm. It is a weighted sum (with weights $0, 1, \ldots, s-1$ and r) of the logarithmic G-S lengths, where the $r - s$ removed vectors are counted as if they had squared G-S lengths $2^{3r}B$.*

Theorem 1. *Gradual B-reduce has switch-complexity $\mathcal{O}(r^3 + r^2 \log(B))$ if we assume that the Gram-Schmidt lengths of $\pi(b_2), \ldots, \pi(b_r)$ are at least 1.*

Proof. The assumption on the Gram-Schmidt lengths implies that the value of P at the end of step 2 is nonnegative. Substituting $s = 0$ gives an upper-bound for P because of lemma 2. Hence $P \leq r^2 \log_{4/3}(2^{3r} B) = \mathcal{O}(r^2(r + \log(B))$ will hold at any time during steps 3 and 4. Because each LLL-switch increases P by at least 1, we can prove the theorem by showing that the progress, P, never decreases during steps 3 and 4.

Removing a vector can only increase P by Lemma 2. The only steps that change G-S lengths are scalings and LLL-switches. Lemma 1 shows that scaling up (step 3b) can only increase G-S lengths and hence can not decrease P. $\square$

2.2 Partial Reduction of a Special Matrix

We want to bound the switch-complexity of B-reducing the rows of the following type of matrix:

$$\begin{pmatrix} & & & & & d_N \\ & & & & \iddots & \\ & & & d_1 & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}$$

where the lower left hand corner is an $r \times r$ identity matrix, $*$ represents any number, and ${d_i}^2 > 2^{((r+1)^2-(r+1))/2} B^{(r+1)}$. In addition we require $|d_i| > 1$ and that d_i has the largest absolute value in its column.

Algorithm 4. *Matrix Reduce*

Begin with the vectors $S = b_1, \ldots, b_r$ which are the rows of the $r \times r$ identity matrix in the bottom left corner, throughout the algorithm we will let s denote the current number of vectors in S and n_{rm} denote the number of removed vectors. Then perform the following:

1. *Add a row and a column. By this we mean expand the corner of the matrix we will deal with, which includes adjoining a new entry for each current row-vector and then inserting the next row-vector (properly truncated). Note that the new vector is inserted as the* first *element in S (this will be used in the proof of lemma 5 below).*

2. *Call Algorithm 3 on S.*

3. *Unless the matrix has been exhausted go back to step 1.*

Example of Algorithm 4

Let

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 1000 \\ 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 1000 & 0 & 0 \\ 1 & 0 & 333 & 460 & 371 \\ 0 & 1 & 665 & -81 & -258 \end{bmatrix}$$

Let H denote the largest absolute value of the entries of M. We choose $B = 10$. Let L be lattice generated by the *rows* of this matrix. In summary, we have

$$r = 2, N = 3, B = 10, H = 1000.$$

Task: B-reduce L. This means finding $b_1, \ldots, b_s \in L$ with the following properties

- $b_1, \ldots, b_s$ is LLL-reduced.
- Any $v \in L$ with $\| v \|^2 \leq B$ is in $\mathrm{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_s)$.

Lemma 7 below states that regardless of the size of H, this task can be done using no more than this many LLL switches: $N(r+1)\log_{4/3}(2^{3(r+1)}B) = 3(2+1)\log_{4/3}(2^{3(2+1)}10) = 9\log_{4/3}(2^9 10)$ which is 267 rounded down. So even if we replaced the non-zero numbers in the last three columns of the example M by million-digit integers we would still have the same switch-complexity. Of course, for larger H the CPU time does increase, since it takes more time to add many-digit integers than it does to add small integers. However, the key point of this section is that the switch-complexity (the bound for the number of LLL switches) depends only on r, N, and B but is *independent of* H.

Example continued: We have $r = 2$, $N = 3$, and $B = 10$.

Initialize: Let $S := (1,0),(0,1),\ \ s := 2,\ \ n_{\mathrm{rm}} := 0$.

1. Add entries and we obtain: $S = (1,0,333),\ (0,1,665)$.

Now add a vector: $S = (0,0,1000),\ (1,0,333),\ (0,1,665)$ and let $s := 3$.

2. Call Algorithm 3. This produces $S = (1,1,-2),\ (-2,1,-1)$. Now $s = 2$ and $n_{\mathrm{rm}} = 1$.

3. There are two more rows/columns left, so go back to step 1.

1. Add entry. To calculate the new entry we need to use entries numbered $1, \ldots, r$. We obtain: $S = (1,1,-2,1\cdot 460 + 1\cdot(-81)),\ (-2,1,-1,-2\cdot 460 + 1\cdot(-81))$.

Now add a vector: $S = (0,0,0,1000),\ (1,1,-2,379),\ (-2,1,-1,-1001)$.

2. Call Algorithm 3. This produces $S = (-2,1,-1,-1)$. Now $s = 1$ and $n_{\mathrm{rm}} = 3$.

3. There is one more row/column left, so go back to step 1.

1. Add entry. $S = (-2,1,-1,-1,-2\cdot 371 + 1\cdot(-258))$.

Now add a vector: $S = (0,0,0,0,1000),\ (-2,1,-1,-1,-2,-1000)$.

2. Call Algorithm 3. This produces $S = (-2, 1, -1, -1, 0)$. Now $s = 1$ and $n_{\mathrm{rm}} = 4$.

3. We used all rows/columns.

Output: $(-2, 1, -1, -1, 0)$.

If $b_1, \dots, b_s$ is the current collection of vectors and n_{rm} is the number of vectors which have been removed, then we define the Progress, P, of the current vectors as:

$$P(b_1, \dots, b_s) = 0 \cdot l_1 + 1 \cdot l_2 + \cdots + (s-1) \cdot l_s + (r+1) n_{\mathrm{rm}} \log_{4/3}(2^{3(r+1)} B)$$

Lemma 3. *In step 2 at least one vector gets removed so that s is never more than $r+1$ in Algorithm 4.*

Proof. If no vector gets removed then the determinant squared must be the same both before and after step 2. But before step 2 the determinant squared is $\geq B^{(r+1)} 2^{((r+1)^2 - (r+1))/2}$ (because of the d_i). After step 2 we have $\| b_s^* \|^2 \leq B$ and $\| b_i^* \|^2 \leq 2^{s-i} \| b_s^* \|^2$ which imply that our determinant squared must be $\leq B^s 2^{(s^2 - s)/2}$. Since the first time step 2 is called we have $s = r+1$ we know that at least one vector must have been removed. But now by repeating this logic we have $s \leq r+1$ for every other time step 2 is called. Therefore every time step 2 is called at least one vector is removed. Also the product of the G-S lengths is the determinant of S. So if we were to keep all vectors when adding a column we would need that $\| b_s^* \|^2 \leq B$. These facts imply that the determinant squared of the starting lattice must be below $B^s 2^{(s^2-s)/2}$ but ${d_i}^2 > B^{(r+1)} 2^{((r+1)^2 - (r+1))/2}$ for all i, so that the determinant of any of the lower left hand corners we take is large enough to guarantee at least one vector is removed. ☐

Corollary 1. *No removed vector (at the time of its removal) can have l_i larger than $\log_{4/3}(2^{3(r+1)} B)$, and when lattice reduction (Algorithm 1) is called no remaining vector has l_i larger than $\log_{4/3}(2^{3(r+1)} B)$.*

The proof is the same as the proof for Lemma 2.

Lemma 4. *The G-S lengths of any vector in Algorithm 4 are always ≥ 1. In other words $l_i \geq 0$.*

Proof. The initial vectors have G-S length ≥ 1 ($d_i \geq 1$ above). So the lemma follows from Observation 1. ☐

Lemma 5. *Progress P never decreases in Algorithm 4.*

Proof. We have already shown that Algorithm 3 will not decrease progress. It remains to show that step 1 will not decrease progress.

Step 1 adds the vector $(0, \dots, 0, d_i)$ to the beginning of $b_1, \dots b_s$. This way the G-S lengths of $b_1, \dots, b_s$ are not changed by the addition of an extra entry. They will only have their weights increased by one in P, which can only increase P (because of lemma 4). The length of $(0, \dots, 0, d_i)$ has no impact on P since it is counted with weight 0. ☐

Lemma 6. *Each LLL switch increases Progress, P, by at least 1.*

This is the same as in the previous section.

Lemma 7. *P is always $\leq N(r+1)(\log_{4/3}(2^{3(r+1)}B)$, so that $P = \mathcal{O}(Nr(r + \log(B)))$.*

Proof. As seen in the previous section when Gradual B-Reducing no l_i can become larger than $\log_{4/3}(2^{3(r+1)}B)$. Thus a remaining vector contributes less to progress than a removed vector. In this application however we do allow a large G-S length but only when the new vector is added in step 1. However by adding this vector as the first vector in our set it contributes nothing to progress and when algorithm 3 is called its size is immediately scaled back down. Thus the progress is always less than it would be if all vectors were removed. □

This implies that the number of LLL switches is bounded by
$N(r+1)\log_{4/3}(2^{3(r+1)}B) = \mathcal{O}(Nr(r + \log(B)))$, since each switch ensures an increase of at least 1.

2.3 New Bounds for Factoring in $\mathbb{Q}[x]$

Now we simply observe that factoring over $\mathbb{Q}$ can be accomplished by B-reducing a matrix with the same format as our special matrix in the previous section.

Notation: Let $f \in \mathbb{Q}[x]$ be a polynomial of degree N, and p a prime such that $f \equiv f_1 \cdots f_r \bmod p$ is the factorization of f in $\mathbb{F}_p[x]$. Let $f \equiv \tilde{f}_1 \cdots \tilde{f}_r$ be the factorization of f mod p^a for some positive integer a, with $f_i \equiv \tilde{f}_i \bmod p^a$ for all i. Let $B := r + 1$.

We will make some minor changes to the All-Coefficients matrix defined in section 1.4.2 to produce a matrix that looks like:

$$\begin{pmatrix} & & & & & p^{a-b_N} \\ & & & & \iddots & \\ & & & p^{a-b_1} & & \\ 1 & & & * & \cdots & * \\ & \ddots & & \vdots & \ddots & \vdots \\ & & 1 & * & \cdots & * \end{pmatrix}$$

where p^{b_i} represents $\sqrt{N}$ times a bound on the ith coefficient of the logarithmic derivative, and we have Hensel lifted high enough to ensure that $p^{a-b_i} > 2^{((r+1)^2-(r+1))/2}B^{(r+1)}\sqrt{N}$ for all i. A B-reduction of this matrix will also solve the recombination problem by a similar argument. Thus the switch complexity of factoring over $\mathbb{Q}$ is bounded by $N(r+1)(\log_{4/3}(2^{3(r+1)}B) = \mathcal{O}(Nr^2)$ (this adjustment allows B to only depend on r).

2.4 Conclusion of Chapter

This new switch-complexity of $\mathcal{O}(Nr^2)$ is an improvement over [10, 8, 3] which has switch complexity $\mathcal{O}(N^2(N + \log(H)))$, where $H = \| f \|_\infty$. Note that our switch complexity is independent of coefficient size.

We tried to create an algorithm that would allow us to bound the LLL switches while not deviating too far from the fastest current implementations like [2].

One might wonder why the central algorithm of this chapter is not presented in more detail, after all its complexity bound is better than prior complexity bounds. The truth is that it is unlikely that this algorithm will be used, because existing implementations perform much better in practice (see the remark in section 1.4.2). Also we are about to present an algorithm with even better switch complexity for which we give full details.

The main purpose of presenting this algorithm was to introduce the most important concepts of the algorithm appearing in the upcoming chapter. We will heavily rely on the concept of Progress, P, for bounding LLL switches, we will also feed in the information gradually as we did in this chapter. The new algorithm will allow us to prove a switch complexity of $\mathcal{O}(r^3)$, independent of both degree and coefficient size.

CHAPTER 3

Factoring with switch complexity independent of degree and coefficient size

3.1 Why go on?

We have just shown that our algorithm from Chapter 2 has the best switch complexity of any factoring algorithm over the rationals. Also the upcoming chapter is long, technical, and difficult to write. So why aren't we content with the new algorithm from chapter 2? There are some problems with the chapter 2 algorithm:

1. **A Small Problem** It doesn't seem like this complexity is sharp since the combinatorial problem in Zassenhaus only depends on r, while this Switch Complexity ($\mathcal{O}(Nr^2)$) depends on N as well.

2. **A Big Problem** It is unlikely that this algorithm will be used because existing algorithms perform much better in practice (see the Remark in section 1.4.2).

   The algorithm in chapter 2 is impractical because the amount of Hensel Lifting required is very large and the algorithm only begins when that level of accuracy is reached. In fact problem 1 with Zassenhaus' algorithm from section 1.2.2 has been made much worse and not better! In fact there is a third problem which is true of all prior algorithms:

3. **Problem with all prior algorithms:** Every existing algorithm requires using just as much Hensel Lifting as the Zassenhaus algorithm even though there are examples where less lifting would be sufficient.

We would like to resolve all three problems with a single new algorithm:

The goal of this chapter is to make an algorithm which is the fastest both in theory and in practice.

To do this we have to find a way of using less Hensel Lifting, and we would like to remove the N from the switch-complexity. Which means that we will have to try solving the problem before very much Hensel

lifting, and we cannot use all N logarithmic derivative coefficients. One key idea is to only use data that is likely to help solve the problem.

3.2 Introducing the Active Determinant

We will present an algorithm that follows the ideas of the previous chapter with the following differences: We use much less hensel lifting and we must skip some of the coefficients of $\frac{g'f}{g}$ in order to obtain a switch-complexity which is independent of N, the degree of f. Which means we need to have a way of deciding when a coefficient/new entry will lead to progress and when a coefficient/new entry will not lead to progress.

Consider the following situation from Algorithm 4 in Section 2.2:

1. We add a new entry to each vector and a new vector to our lattice (Step 1 in algorithm 4)

2. We run Gradual B-Reduce (Algorithm 3) to increase progress which does the following:

   (a) LLL reduction during calls to algorithm 2

   (b) Removes vectors in algorithm 2 which are no longer needed (perhaps also increasing progress).

Now $b_1, \ldots, b_s$ is B-reduced, and if we were to run LLL again immediately we would accomplish nothing (no switches would be made). Suppose that the next time we run Step 1 in Algorithm 4 the new entry which we add to $b_1, \ldots, b_s$ is already very small. In this case running B-reduce/LLL will still not make many switches (progress) because all of the new entries are so small that the basis has a good chance of staying LLL reduced (the G-S lengths are not altered by much). So we need to know when extra information (a new entry) will lead to progress and when it won't.

Definition 7. *Let $b_1, \ldots, b_s$ be the active vectors and b_i^* the ith G-S vector, then we call $AD = \prod \parallel b_i^* \parallel$ the Active Determinant.*

This is a measure of how much usable information is stored in the lattice currently (in the form of G-S length). AD is independent of LLL since LLL switches don't change the product of the involved G-S lengths. So the scenario of adding entries (or any other action we take) will now look like this:

1. We only add a new entry (and make some action) if it increases AD by a sufficient amount.

2. We run LLL which is guaranteed to increase Progress because of 1.

3. We remove vectors (which decreases AD by a bounded amount per removed vector).

4. Repeat: We must increase AD again before LLL can make more progress.

This is the basic loop of the new algorithm. Termination and complexity bounds can be derived if we ensure that:

1. AD is bounded

2. The amount we choose to increase AD by in step 1 is at least 2^r more than the amount removed in step 3.

3.3 How to read this chapter

This algorithm is highly recursive and the proof of its complexity rather technical. We present the algorithm divided into its 10 procedures.

So we have presented each procedure as its own independent section. Each procedure's section is divided into several subsections:

1. A list of variables used in the procedure with meanings and references.

2. Each section then has the procedure written out in full detail.

3. We list the properties which should be true at the moment the procedure is called. We assume these in the proofs.

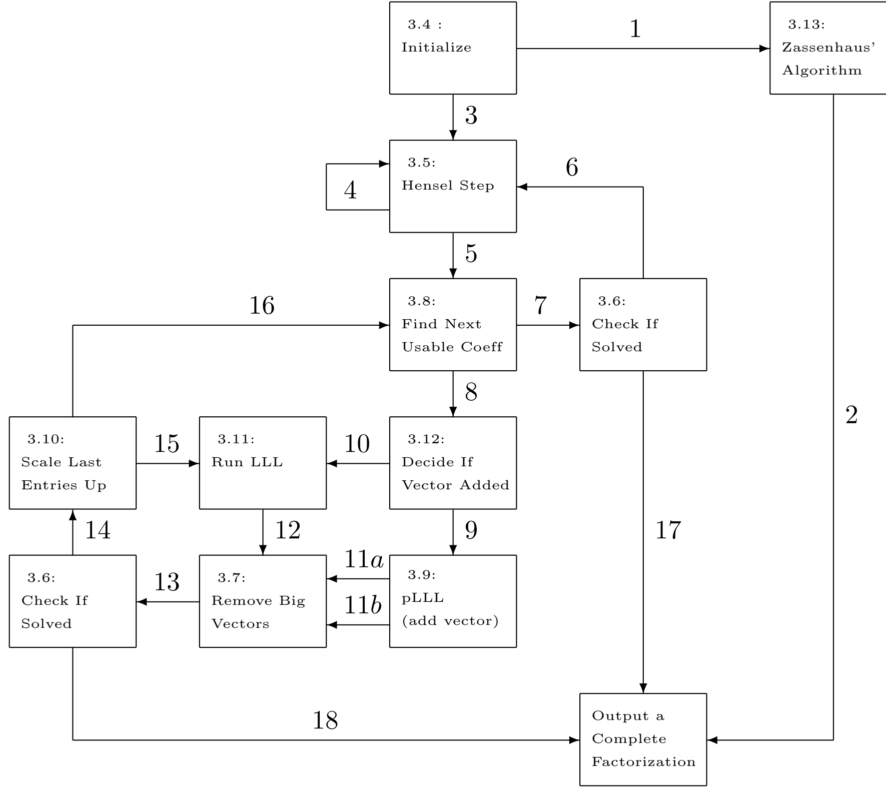
4. We then prove the properties which should be true at the moment the procedure is terminated and another procedure is called. This is the subsection which contains the meat of our proofs.

5. Finally we present some notes on the complexity of the section as it related to the entire algorithm. This should include comments on how many times this procedure is called from other procedures and how often this procedure calls other procedures.

We provide a summary sheet that contains the flow chart and the 14 properties which should be proved at every output/exit arrow. Properties 1 through 11 should be proved verbatim and will remain true throughout the entire algorithm. The other three properties are more procedure specific and the . . . on the summary sheet will be filled in for each specific procedure. The heart of our claims on switch-complexity are summarized in **8.** $n_{\text{switches}} \leq P$ and the bound on P proved in section **3.6**. The later properties illustrate the effect of a given procedure on AD and $\parallel b_i^* \parallel$. While properties 1-7 are global properties which are needed in many of the proofs.

3.3.1 Summary Sheet

The r^3 Algorithm:



For each procedure/section prove the following 14 properties:

1. $s < \lfloor \frac{3r}{2} \rfloor$	$s = r + n_{\text{good}} - n_{\text{rm}}$ is the dimension of L
2. $n_{\text{good}} \leq 3r + 2$	n_{good} counts 'good' coefficients (Exit 11a)
3. $n_{\text{bad}} \leq 3r^2 - 2r + 1$	n_{bad} counts 'bad' coefficients (Exit 11b)
4. $n_{\text{novec}} \leq 3r + 2$	n_{novec} counts 'no vector' coeffs (Exit 10)
5. $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r + 1$	n_{rm} counts vectors removed in **3.7**
6. $n_{\text{scales}} \leq 3r + 2$	n_{scales} counts successful scalings (Exit 15)
7. $n_{\text{entries}} \leq 3r^2 + 5r + 5$	n_{entries} counts entries in b_i, i.e. $L \subseteq \mathbb{Z}^{n_{\text{entries}}}$
8. $n_{\text{switches}} \leq P$	n_{switches} counts LLL switches throughout
9. $P^{\text{out}} \geq P^{\text{in}}$	Progress, P, is how we bound n_{switches}
10. $W \subseteq \pi(L)$	W is the solution lattice in $\mathbb{Z}^r$
11. $\parallel \pi^{-1}(v_g) \parallel^2 \; \leq \; r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2$ $\leq r + 1$ for any irreducible factor g	This is how we ensure $W \subseteq \pi(L)$
12. $\max\{\parallel (b_i^{\text{out}}))^* \parallel\} \leq 2^{\cdots}$	We bound the G-S length of any b_i
13. $\text{AD}^{\text{out}} \leq 2^{\cdots}$	AD is always bounded by something
14. $\text{AD}^{\text{out}} \geq \ldots \text{AD}^{\text{in}}$	Also AD increases after most procedures

3.3.2 Notations

1. In order to show change in any variable we use the convention of x^{out} to represent the value of any variable, x, at the conclusion of the sub-procedure and x^{in} to be the value of the variable at the input.

2. Let $b_1^*, \ldots, b_s^*$ be the Gram-Schmidt Orthogonalization (G-S) of $b_1, \ldots, b_s$.

3. Let $l_i := \log_{\sqrt{4/3}}(\| b_i^* \|) = \log_{4/3}(\| b_i^* \|^2)$. Also let $l_{i,2} := \log_2(\| b_i^* \|)$.

4. L is $\text{SPAN}_{\mathbb{Z}}\{b_1, \ldots, b_s\}$. The phrase 'Our Lattice' refers to L.

5. AD:=det$(L) :=\| b_1^* \| \cdots \| b_s^* \|$.

6. In all sections but **3.9(pLLL)** we will use the following formula for progress:

$$P := 1 \cdot l_1 + \cdots + (s) \cdot l_s + \frac{3r}{2} \cdot n_{\text{bad}} + \frac{3r}{2} \cdot 2r \log_{\sqrt{4/3}}(2) \cdot n_{\text{rm}}$$

7. In **3.9** we use a variant of this formula to deal with the probationary vector (the vector with $l_{i,2} > 2r$). During **3.9** if there is an $l_{j,2} > 2r$, then:

$$P := 1 \cdot l_1 + \cdots + (j-1) \cdot l_{j-1} + (j) \cdot l_{j+1} + \cdots + (s-1) \cdot l_s + \frac{3r}{2} \cdot n_{\text{bad}} + \frac{3r}{2} \cdot 2r \log_{\sqrt{4/3}}(2) \cdot n_{\text{rm}} + (j-1)$$

   Otherwise use the formula for P from item 6.

   Essentially when $l_{i,2} > 2r$ the formula is the formula from item 6 with l_j removed, and $j-1$ added.

8. f is the polynomial to be factored, p a prime chosen so that f is squarefree mod p. Let $f_1, \ldots, f_r \in \mathbb{Z}_p[x]$ be the p-adic factors of f. Let $\tilde{f}_1, \ldots, \tilde{f}_r \in \mathbb{Z}[x]$ be approximations of $f_1, \ldots, f_r$ of accuracy a. This means that $\tilde{f}_i \equiv f_i \bmod p^a$, which implies that $f \equiv \tilde{f}_1 \cdots \tilde{f}_r \bmod p^a$.

9. Hensel Lifting is the process of increasing the accuracy of $\tilde{f}_1, \ldots, \tilde{f}_r$ as approximations of $f_1, \ldots f_r$. (Increasing a). This process is similar to a Newton Step; it doubles the precision a.

10. We use the notation $(v)_i$ to represent the ith entry of any vector v. Also for simplicity $(v)_{-1}$ will denote the final entry of v.

11. Throughout the chapter we use v_g to represent the 0-1 vector corresponding with a monic irreducible rational factor g of f. More precisely: If $g|f$ and g irreducible in $\mathbb{Z}[x]$ then there exists a vector $v_g \in \{0,1\}^r$ with $g \equiv (f_1)^{(v_g)_1} \cdots (f_r)^{(v_g)_r} \bmod p^a$.

12. The word mods denotes the symmetric remainder. n mods p^a returns the number, $\tilde{n}$ in the congruence class of n modulo p^a with $\tilde{n} \in (-\frac{p^a}{2}, \frac{p^a}{2}]$.

13. We use $\text{CLD}_c^a(g) := \text{Coefficient}(\frac{g'f}{f}, x^c)$ mods p^a, to represent the cth Coefficient of the Logarithmic Derivative times f, for any factor g of f. Note that $\text{CLD}_c^a(f_i)$ can be computed from $\tilde{f}_i$ because the accuracy of $\tilde{f}_i$ is a.

14. $\text{CB}(c)$ is chosen so that $2^{\text{CB}(c)} \geq |\text{CLD}_c^a(g)|$ for any rational factor, g, of f. There is a bound for $\text{CB}(c)$ in [3], namely $\text{CB}(c) \leq \lceil \log_2(2^{N-1} \cdot N \cdot \| f \|_2) \rceil$. Note: we write $\leq$ because a better bound is possible.

15. $d := \lceil \log_2(\sqrt{3r^2 + 5r + 5}) \rceil$ is chosen so that $2^d \geq \sqrt{n_{\text{entries}}}$ throughout the algorithm. n_{entries} will be defined in **3.4**.

16. $\text{Avail_Bits}(c) := \lceil \log_2(\frac{p^a}{2^{CB(c)+d}}) \rceil$, tells us how many bits of information above the coefficient bound our algorithm can use. If this turns out to be insufficient then we can increase it by Hensel Lifting.

17. We use $\psi_c^a(v)$ to represent $\displaystyle\sum_{i=1}^{r} (v)_i \text{CLD}_c^a(f_i)$. Note that $\psi_c^a(v_g)$ is congruent to $\text{CLD}_c^a(g)$ mod p^a but need not be equal because CLD_c^a always returns a number which is reduced mods p^a where as ψ_c^a might not.

18. IB should be an experimentally determined bound such that if $a < \text{IB}$ then it becomes unlikely that the algorithm can terminate without additional calls to **3.5(Hensel Step)**. For now we choose IB := the value for a for which $\sum \text{Avail_Bits}(c)$ is approximately $\frac{r^2}{8}$.

19. Let $W \subseteq \mathbb{Z}^r$ be the lattice generated by the v_g. In other terms: If f factors completely over $\mathbb{Q}$ as $g_1 \cdots g_k$ then define $W := \text{SPAN}_{\mathbb{Z}}\{v_{g_1}, \ldots, v_{g_k}\}$ where v_{g_i} is the 0-1 vector corresponding with g_i, see item 11.

20. We use $\pi(b_i)$ to represent the projection of b_i onto its first r entries. Also $\pi(L) = \text{SPAN}_{\mathbb{Z}}\{\pi(b_1), \ldots, \pi(b_s)\}$.

21. Let $\pi^{-1}(v_g)$ denote a shortest vector in the pre-image of v_g. By design of the algorithm we have $\pi^{-1}(v_g) = (v_g, \frac{\text{CLD}_{c_1}^a(g)}{\text{TS}_1}, \ldots, \frac{\text{CLD}_{c_{\text{tc}}}^a(g)}{\text{TS}_{\text{tc}}})$. The notation TS_i and tc will be introduced in **3.10(Scale Up)** and **3.4(Initialize)** respectively.

22. Just like in [6] the aim is to keep making the lattice $\pi(L)$ smaller during the algorithm until eventually $\pi(L)$ will be W at termination.

23. Is_Zero_One_Basis$(b_1, \ldots, b_s)$ decides if there is a 0-1 basis for $\pi(L)$.

24. Attempt_Reconstruction tries to reconstruct a rational factor of f from the $\tilde{f}_i$ if $\pi(L)$ has a 0-1 basis. The procedure simply multiplies the $\tilde{f}_i$ mods p^a and does a trial division of f. (See step 10 of algorithm 2.2 in [6])

3.3.3 Some useful Facts

Fact 2. *If $g \in \mathbb{Q}[x]$ is an irreducible factor of f, then* $\frac{Coeff_c(g)}{2^{CB(c)+d}} \leq \frac{1}{\sqrt{3r^2+5r+5}}$

Observation 2. *It is important to distinguish $CLD_c^a(g)$ and $\psi_c^a(v_g)$, the true $CLD_c^a(g)$ is reduced mods p^a where as ψ_c^a might not be. So when we add new coefficients we need to add a vector of the form $(\vec{0}, p^a)$ to our lattice in order to verify property **11** from the summary sheet.*

NOTE: Property **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r+1$ for any irreducible factor g, will hold true if each new entry we add to $\pi^{-1}(v_g)$ is of size $\leq \frac{1}{\sqrt{3r^2+5r+5}}$ because of property **7.** $n_{\text{entries}} \leq 3r^2+5r+5$.

Lemma 8. *If $r > 10$ and there is a B-reduced sequence of $s \leq 3r/2$ elements (and $B = r+1$) then the AD of this sequence is $\leq 2^{r^2}$, and $\| b_i \| \leq 2^{r-1} \leq 2^r$ for all i.*

Proof. LLL reduced sequences satisfy the property $\| b_i^* \|^2 \leq 2 \| b_{i+1}^* \|^2$ for any i. A B-reduced sequence has the final element with $\| b_s^* \|^2 \leq B \leq 2^{r/2-1}$ if $r \geq 10$. So $\| b_i^* \| \leq 2^{(s-i)/2+r/2-1}$. When $s < 3r/2$ we see $\| b_1 \| \leq 2^{r-1}$. Thus the product of G-S lengths is $\leq B^{3r/4} 2^{(3r/2-1)3r/8} \leq 2^{r^2}$. The $\| b_i \| \leq 2^{r-1}$ follows from (1.7) in LLL. □

3.3.4 The basic idea of the following algorithm

In **3.6(Check If Solved)** we see that the algorithm can only terminate if $\pi(L) = W$. We start in **3.4(Initialize)** with $\pi(L) = \mathbb{Z}^r$ and $s = r$. The aim is now to append entries to the basis $b_1, \ldots, b_s$ so that the vectors $v_{g_1}, v_{g_2}, \ldots$ stay smaller than B, while the vectors v with $\pi(v) \notin W$ grow large so that Gradual B-reduce (algorithm 3 in the previous chapter and arrows 12, 13, 14 and 15 in this chapter) can perhaps remove them. Each time we add (or scale) an entry we increase AD, the determinant of $b_1, \ldots, b_s$. If AD increases by a sufficient amount it becomes inevitable that there is a G-S length large enough for B-reduce to remove it; thereby bringing $\pi(L)$ closer to W. The tricky problem is that each time we add an entry we also need to add a vector (of the form $(0, \ldots, 0, *)$). So we need to ensure not only that vectors are

removed every once in a while, but also that, at least in the long run, more vectors are removed then are added. This is done by ensuring that each removed vector lowers the determinant by less than the amount the determinant increases when we add a vector. We can make sure of this by providing an upper bound for the size of a removed vector and a lower bound for the size of an inserted vector. Then an upper bound for AD will ensure that the number of removed vectors must, in the long run, outnumber the number of added vectors.

So in **3.9(pLLL)** we add a new entry and vector just like in the example of algorithm 4 from section 2.2. We then run LLL reduction and make sure that the vectors are small again before we remove them (if there is a vector which is not small enough we call this a **Bad Vector**).

Then we remove any vectors whose G-S lengths are $> B$ in **3.7(Remove Vectors)**. Next **3.10(Scale Up)** is the procedure by which we squeeze every bit of information we can get from the new entry. This is why there is a loop in the lower left hand corner of the flow chart. In fact this loop is basically the Gradual B-Reduce algorithm from section 2.1.

In the odd case that the size of the new vector $(\vec{0}, p^a)$ is much much larger than the new entries, we might be able to remove it without using any LLL switches. This is the **3.12(No Vector Added)** procedure.

3.4 Initialize

3.4.1 Variables used in this Procedure:

- r represents the number of local factors.
- $f_1, \ldots, f_r$ are the local factors, their images in $(\mathbb{Z}/p\mathbb{Z})[x]$ are $\tilde{f}_1, \ldots, \tilde{f}_r$ irreducible with $\tilde{f}_1 \cdots \tilde{f}_r \equiv f \mod p$.
- b_i is the ith vector in the current basis of our lattice L. Most of the work of the algorithm is done by altering the b_i.
- n_{entries} will be a counter for the number of entries in each b_i. It will change when we add a new entry to each b_i, which happens only in **3.9** step 1 or **3.12** step 10e.
- We also use tc which is just $n_{\text{entries}} - r$, but provides a convenient index for the history of our coefficients. (For instance I might need to refer to the total scaling done to the fifth coefficient we encountered, the $(r+5)^{\text{th}}$ entry in each vector. This is stored as TS_5.)
- a represents our p-adic accuracy at the moment. It only changes during **3.5(Hensel Step)**, and when a new coefficient is selected we store the p-adic accuracy under a_{tc} for reference in proofs.

- $B = r+1$ is an upper bound for the length of $\pi^{-1}(v_g)$ throughout the algorithm, for any factor $g \in \mathbb{Z}[x]$ of f. (Provided property **11** holds throughout.)

- d is chosen so that $2^d \geq \sqrt{n_{\text{entries}}}$ throughout the entire algorithm. It's used to make sure we don't over scale in **3.12(No Vector Added)** and **3.10(Scale Up)**. It also contributes to the amount of information we consider sufficient in **3.8(Find Next Coeff)**.

- n_{good} counts the number of good vectors we have encountered so far and only changes during **3.9(pLLL)**.

- n_{bad} counts the number of bad vectors we have encountered so far and only changes during **3.9**.

- n_{novec} counts the number of No_Vector_Added coefficients (coefficients for which no vector needed to be added as decided in **3.12(No Vector Added)**) we have encountered so far and only changes during **3.12**.

- n_{rm} counts the number of vectors which have been removed during **3.7(Remove Vectors)**.

- n_{scales} counts the number of successful scalings have taken place in **3.10**.

- s is the current number of vectors (b_i) in the algorithm at the moment. This will be $n_{\text{good}} + r - n_{\text{rm}}$ except during **3.9** where $s = n_{\text{good}} + r - n_{\text{rm}} + 1$ during the LLL run there. This number only changes during **3.9** and **3.7**.

- n_{switches} counts the number of LLL switches which take place in the algorithm, this is the number we are most concerned with bounding. It changes during **3.11(LLL)** and **3.9(pLLL)**.

- ϵ is a constant and is chosen somewhat arbitrarily as 0.1. It is used during some proofs in **3.12** and **3.10**.

3.4.2 Procedure

1. Find r and $f_1, \ldots, f_r$ irreducible mod p, with $f_1 \cdots f_r \equiv f \bmod p$. (See [5])

2. If $r < 10$ then exit this procedure and call **3.13(Zassenhaus)** (Exit 1)

3. $b_1, \ldots, b_r := e_1, \ldots, e_r \in \mathbb{Z}^r$

4. $s := r$

5. $a := 1$

6. $d := \lceil (1/2)\log_2(3r^2 + 7r + 10) \rceil$

7. $n_{\text{entries}} := r$

8. $B := r + 1$

9. $n_{\text{novec}}, n_{\text{good}}, n_{\text{bad}}, n_{\text{rm}}, n_{\text{scales}} := 0, 0, 0, 0, 0$

10. $\text{tc} := n_{\text{good}} + n_{\text{bad}} + n_{\text{novec}}$

11. $\epsilon := .1$

12. $n_{\text{switches}} := 0$

13. Exit This Procedure and Call **3.5(Hensel Step)** (Exit 3)

3.4.3 What This Procedure Does:

- This is where factoring mod p takes place. One of the key steps in Zassenhaus' algorithm.
- Since Zassenhaus' algorithm has an exponential running time in r it will be efficient to call **3.13(Zassenhaus)** for sufficiently low r.
- If r is large enough we will define some starting variables to be used and altered throughout the algorithm.
- This procedure initializes $L = \text{SPAN}_{\mathbb{Z}}\{b_1, \ldots, b_s\}$ for the first time with $L = \mathbb{Z}^r$. $b_1, \ldots, b_s$ only change during the procedures: **3.11(LLL)**, **3.9(pLLL)**, **3.12(No Vector Added)**, **3.10(Scale Up)**, and **3.7(Remove Vectors)**. While **3.11** does alter the b_i it doesn't alter L.

3.4.4 Properties True at Exit 1

- $r < 10$ so Zassenhaus will be efficient.

3.4.5 Properties True at Exit 3

- $L = \mathbb{Z}^r$, AD $= 1$, and $\max(l_i) = 0$.
- Our basis $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1).
- Properties **1, 2, 3, 4, 5, 6,** and **7** are all true by definition.

- **9.** $P^{\text{out}} \geq P^{\text{in}}$ doesn't apply. However since $n_{\text{bad}} = 0, n_{\text{rm}} = 0$, and $l_i = 0$ for all i, we know $P = 0$. Which confirms **8.** $n_{\text{switches}} \leq P$ since $n_{\text{switches}} := 0$.

- **10.** $W \subseteq \pi(L)$ since $v_g \in \mathbb{Z}^r = L$ for every factor of f. W is defined in item 19 in Section 3.3.2.

- **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r + 1$ for any irreducible factor g is true because $\pi^{-1}(v_g) = v_g$ at the moment, and v_g is a 0-1 vector in $\mathbb{Z}^r$.

- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^0$ and **13.** $\text{AD}^{\text{out}} \leq 2^0$ are mere observations.

- Property **14** doesn't apply since there is no AD^{in}.

3.4.6 Complexity Notes:

- This procedure is only called once.

- Exit 1 happens at most once.

- Exit 3 happens at most once.

- The CPU cost of this procedure is dominated by Factoring mod p. (see [4] for complexity results)

3.5 Hensel Step

3.5.1 Variables used in this Procedure:

- $f_1, \ldots, f_r$ are the p-adic factors of f, $\tilde{f}_1, \ldots, \tilde{f}_r$ are approximations of the f_i with accuracy a (defined in **3.4**). It should always be true that $\tilde{f}_1 \cdots \tilde{f}_r \equiv f \bmod p^a$.

- IB is a level of p-adic accuracy chosen so that there is a good chance of solving the problem when $a \geq \text{IB}$. (See Section 3.3.2)

- c will be the current coefficient of $\frac{g'f}{g}$ that we will explore. It will update in **3.8(Find Next Coeff)** and **3.10(Scale Up)**, and reset whenever we need more Hensel Lifting.

3.5.2 Procedure

1. Improve the accuracy of $\tilde{f}_1, \ldots, \tilde{f}_r$ to $2a$. So now: $\tilde{f}_1 \cdots \tilde{f}_r \equiv f \bmod p^{2a}$.

2. $a := 2a$

3. If $a < \text{IB}$ then exit this procedure and call **3.5(Hensel Step)** (Exit 4)

4. $c := 0$

5. Exit this procedure and call **3.8(Find Next Coeff)** (Exit 5)

3.5.3 What This Procedure Does:

This procedure uses Hensel Lifting to increase the p-adic accuracy of our local factors of f by a factor 2. The higher the accuracy, a the more information (Avail_Bits) we have for solving the factorization. It was noted in [3] that if the local factors are known to sufficient accuracy (we give this accuracy, called n_{hensel}, in **3.8**) then the all-coefficients matrix (see Section 2.3) will contain enough information to solve the entire problem. So n_{hensel} is a bound for a, meaning that once a reaches n_{hensel} we can be sure that the algorithm will not call Hensel lifting again.

What's new compared to chapter 2 is that we do only one Hensel Step at a time, rather than lifting to $p^{n_{\text{hensel}}}$ at the beginning. In many examples Hensel Lifting is the practical bottleneck of the algorithm. So by beginning our search early, we will minimize this practical bottleneck. This Chapter actually shows that this early checking doesn't hurt the complexity, which means we have resolved the key practical problem from section 1.2.2! We actually make significant practical gains when the polynomial is irreducible or has a single large factor.

$$\begin{aligned} \text{IB} &= \text{probably sufficient Hensel lifting} \\ n_{\text{hensel}} &= \text{provably sufficient Hensel lifting} \end{aligned}$$

The IB was suggested in the additional comments of van Hoeij's original paper. If we haven't lifted to p^{IB} we lift again by calling **3.5(Hensel Step)** (Exit 4). Once IB has been reached we always move on by calling **3.8(Find Next Coeff)** (Exit 5).

3.5.4 Properties true of the Input

- To confirm these check the properties at exits 3, 4, and 6 from **3.4(Initialize)**, **3.5(Hensel Step)**, and **3.6(Check If Solved)** respectively.
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1).
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$

3.5.5 Properties true at exit 4

No variables other than a and c changed and so

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \le 2^r$
- **13.** $\text{AD}^{\text{out}} \le 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$

all remain true.

3.5.6 Properties true at exit 5

No variables other than c and a changed and no vectors changed so the input properties all remain true.

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \le 2^r$
- **13.** $\text{AD}^{\text{out}} \le 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$
- Also $p^a \ge p^{\text{IB}}$. (See 3.3.2.)

3.5.7 Complexity Notes:

- This procedure can only be called $\le \log_2(n_{\text{hensel}})$ times (see section 3.8.3), including the $\log_2(\text{IB})$ times that it calls itself (Exit 4). Although the CPU time is always dominated by the final call.
- This procedure calls itself $\log_2(\text{IB})$ times (Exit 4).
- This procedure is called 1 time from **3.4(Initialize)** (Exit 3).
- Since there are three inputs (two well known already) and we've bounded the total number of calls, then the number of calls from **3.6(Check If Solved)** (exit 6) is bounded by $\log_2(n_{\text{hensel}})-1-\log_2(\text{IB})$.
- There are only two exits and Exit 4 is called $\log_2(\text{IB})$ times, so exit 5 (to **3.8(Find Next Coeff)**) is called at most $\log_2(n_{\text{hensel}}) - 1 - \log_2(\text{IB})$ times.
- The total complexity of all calls to this procedure is dominated by the Hensel Lifting in the final call to this procedure.

3.6 Check If Solved

3.6.1 Variables used in this procedure

- $b_1, \ldots, b_s$, the current basis of L.
- Is_Zero_One_Basis is a sub-procedure for deciding if there is a 0-1 basis of $\pi(L)$ and what it is:
    1. Partition $\{1, \ldots, r\}$ with i, j in the same equivalence class if $(v)_i = (v)_j$ for all $v \in \{b_1, \ldots, b_s\}$.
    2. If the number of classes $= s$ then rref($\pi(b_1), \ldots, \pi(b_s)$) is a 0-1 basis.
- Attempt_Reconstruction is the same procedure from van Hoeij algorithm 2.2 step 10.
- $\tilde{f}_1, \ldots, \tilde{f}_r$ and f are used in Attempt_Reconstruction.

3.6.2 Procedure

1. If called from **3.7(Remove Vectors)** then
    (a) If Is_Zero_One_Basis($b_1, \ldots, b_s$) then
        i. If Attempt_Reconstruction succeeds then exit the algorithm and **Output Factorization** (Exit 18)
        ii. Exit this procedure and call **3.10(Scale Up)** (Exit 14)
    (b) Exit this procedure and call **3.10** (Exit 14)
2. If called from **3.8(Find Next Coeff)** then
    (a) If Is_Zero_One_Basis($b_1, \ldots, b_s$) then
        i. If Attempt_Reconstruction succeeds then exit the algorithm and **Output Factorization** (Exit 17)
        ii. Exit this procedure and call **3.5(Hensel Step)** (Exit 6)
    (b) Exit this procedure and call **3.5(Hensel Step)** (Exit 6)

3.6.3 What This Procedure Does:

This procedure checks if the combinatorial problem is solved, and is called in two places. The only difference in the two if statements is which Procedure gets called in case the problem is still unsolved. (See flow chart on the summary page.)

Definition 8. *Let $g \in \mathbb{Z}[x]$ be a factor of f. We say that a is large enough to reconstruct g from its modular image if and only if $g = (g \text{ mods } p^a)$. In other words, when p^a is larger than $2 \cdot \| g \|_\infty$.*

Unlike other algorithms we already attempt factorization with p^a smaller than the bound mentioned in Zassenhaus' algorithm (Section 1.2). So it is possible that some but not all factors can be reconstructed. Note that in the case where all but one factor is reconstructible the remaining factor can be found by division rather than reconstruction from its modular image.

We've decided to explain this procedure early in the chapter for the benefit of readers who are unfamiliar with other factoring algorithms. To understand what is really happening in van Hoeij style factoring algorithms you must understand the following lemma which is a small variation of a statement in van Hoeij's paper:

Lemma 9. *Assuming **10.** $W \subseteq \pi(L)$, this procedure has an output if and only if $\pi(L) = W$ and a is large enough to reconstruct all, or all but one, of the irreducible factors of f in $\mathbb{Z}[x]$. Further if $\pi(L) = W$ then there is a 1-1 correspondence between the reduced echelon basis of $\pi(L)$ and the set of irreducible factors of f in $\mathbb{Z}[x]$.*

Proof. If $\pi(L) = W$ then a reduced row echelon basis of $\pi(L)$ is the set $v_{g_1}, \ldots, v_{g_k}$ by the uniqueness of RREF, so the algorithm will find a 0-1 basis and it will produce an output if we have Hensel Lifted enough to reconstruct the actual factors. Note that if $\dim(L) = 1$ (i.e. $s = 1$) then we have proved that f is irreducible even if a is too low to reconstruct f from its modular image.

If this procedure produces an output then the 0-1 basis of $\pi(L)$ corresponds with some factorization of f over $\mathbb{Z}$. But because $W \subseteq \pi(L)$ we know that v_{g_i} is in the span of our 0-1 basis of $\pi(L)$, and thus we must have that $\pi(L) = W$ by unique factorization in $\mathbb{Z}[x]$.

W was defined so that its reduced row echelon basis corresponds with a complete irreducible factorization of f. □

In order to use this lemma we need to know that the input satisfies **10.** $W \subseteq \pi(L)$. In order for this procedure to eventually terminate we need $\pi(L)$ to approach W. Which requires vectors $v \in L$ with $\pi(v) \notin W$ to become large (so that **3.7(Remove Vectors)** can remove them).

Every time **3.7(Remove Vectors)** actually removes vectors $\pi(L)$ gets a little bit closer to W. (W remains a subset of $\pi(L)$ using Fact 1 in section 1.3.1.)

To make the $v \in L$ with $\pi(L) \notin W$ large, we keep adding entries to b_i that are small for $\pi^{-1}(v_g)$ and possibly large for $v \in L$ with $\pi(v) \notin W$.

The actual structure of this procedure is as follows: The sub-procedure Is_Zero_One_Vector is a fast way to determine if a 0-1 basis for $\pi(L)$ exists and what it is if it does. Next we only try reconstructing the true factors if we have first found a 0-1 basis for $\pi(L)$.

3.6.4 Properties true of the Input

- To confirm these properties look at exits 7 and 13, from **3.8(Find Next Coeff)** and **3.7(Remove Vectors)** respectively.
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$

3.6.5 Properties true at Exits 17 and 18

At the time of output we have:

Theorem 2. *$n_{switches} \leq P < 68r^3$. Thus the number of LLL switches used throughout the entire algorithm is bounded by $68r^3$.*

Proof. Lemma 14 in Section 3.9.7 shows that $n_{\text{good}}^{\text{in}} \leq 3r + 1$ (and $n_{\text{good}}^{\text{out}} \leq 3r + 2$). The same proof can be used here to show $n_{\text{good}} \leq 3r + 1$. This provides a better bound for $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r$. Similarly the proof of Lemma 13 in 3.9.6 can be used to show $n_{\text{bad}} \leq 3r^2 - 2r$. Also we know that **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$, so plugging these into the formula for P (section 3.3.2 item 6) we get $P^{\text{in}} \leq 67.8r^3$. We then use **8.** $n_{\text{switches}} \leq P$, see 3.6.4. □

Theorem 3. *The output is a complete irreducible factorization of f over the rationals.*

Proof. Follows from lemma 9. □

3.6.6 Properties true at Exits 6 and 14

None of the vectors or any of the variables have changed so the input properties remain true.

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^r$

- **13.** $\text{AD}^{\text{out}} \leq 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$

3.6.7 Complexity Notes:

- **3.5(Hensel Step)** (Exit 6) is called $\leq \log_2(n_{\text{hensel}}) - 1 - \log_2(\text{IB})$ times (see section **3.5**).
- Terminating after **3.8(Find Next Coeff)** (Exit 17) happens at most 1 time.
- The number of calls from **3.8** (Exit 7) is therefore bounded by $\log_2(n_{\text{hensel}}) - \log_2(\text{IB})$.
- This procedure will be called from **3.7(Remove Vectors)** (Exit 13) $n_{\text{novec}} + n_{\text{scales}} + n_{\text{good}} + n_{\text{bad}}$ times (see section **3.7**).
- Terminating after **3.7** (Exit 18) happens at most 1 time.
- So **3.10(Scale Up)** (Exit 14) is called $n_{\text{novec}} + n_{\text{scales}} + n_{\text{good}} + n_{\text{bad}} \leq 3r^2 + 7r + 6$ times or $n_{\text{novec}} + n_{\text{good}} + n_{\text{bad}} + n_{\text{scales}} - 1$ times.
- Thus the total number of calls to this procedure (from either input) is $\leq 3r^2 + 7r + 6 + \log_2(n_{\text{hensel}}) - \log_2(\text{IB}) - 1$ times (still can only terminate once).

3.7 Remove Vectors

3.7.1 Variables used in this Procedure

- s is the current number of vectors in the active basis of L, it only changes in **3.9(pLLL)** and this procedure. We decrease s by one every time we remove a vector (once per loop).
- b_s^* is the final vector in the G-S orthogonalization of $b_1, \ldots, b_s$. If $\parallel b_s^* \parallel$ is larger than B then $\pi(b_s)$ is not needed to ensure **10.** $W \subseteq \pi(L)$ (see Fact 1 in section 1.3.1).
- n_{rm} is only changed in this procedure and counts how many vectors we have removed so far.

3.7.2 Procedure

1. While $\parallel b_s^* \parallel^2 > B$ do
   (a) Remove the last vector, b_s
   (b) $s := s - 1$;
   (c) $n_{\text{rm}} := n_{\text{rm}} + 1$;

od;

2. Exit this procedure and call **3.6(Check If Solved)** (Exit 13)

3.7.3 What This Procedure Does:

This procedure simply removes the final vector until the new final vector has squared G-S length $\leq B = r+1$. Each time this happens s is decreased and n_{rm} increased. The output of this step is actually B-Reduced, since the input is LLL reduced.

Fact 1 in 1.3.1 states that if the final vector in $b_1, \ldots, b_s$ has squared G-S length $> B$ and $v \in \text{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_s)$ with $\| v \|^2 \leq B$ then v is still an element of $\text{SPAN}_{\mathbb{Z}}(b_1, \ldots, b_{s-1})$. Since the basis vectors of W (the v_{g_i}) have square length $\leq B$, it follows that **10.** $W \subseteq \pi(L)$ remains true during this procedure.

3.7.4 Properties true of the Input

- To confirm these properties check **3.9(pLLL)** (both exit $11a$ and $11b$) and **3.11(LLL)** (Exit 12).
- $b_1, \ldots, b_s$ is LLL reduced but not B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^{2r}$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{\frac{3}{2}r^2 + \frac{5}{2}r - 1}$

3.7.5 Properties true at Exit 13

First we need to present an important theorem:

Theorem 4. *The only procedure where AD is decreased is **3.7(Remove Vectors)**. AD can decrease by no more than a factor* $2^{2r \cdot \triangle(n_{rm})}$, *where* $\triangle(n_{rm}) = n_{rm}^{out} - n_{rm}^{in} = s^{in} - s^{out}$.

Proof. There are only five procedures which alter any vectors **3.12(No Vector Added)**, **3.9(pLLL)**, **3.11(LLL)**, **3.7(Remove Vectors)**, and **3.10(Scale Up)**. None of these but **3.7** decreases AD and the proof of this is in each procedure's subsection under the proof of Property **14**.

Observe that the input of **3.7** has the property **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^{2r}$ and each removed vector divides AD by its G-S length. So the only decrease in AD throughout the entire algorithm comes only when n_{rm} is increased and no more than a factor 2^{2r} per removed vector. $\square$

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1) since only final vectors with squared G-S length $> B$ were removed (uses Fact 1 in 1.3.1).
- s certainly did not increase so **1.** $s < \lfloor \frac{3r}{2} \rfloor$ remains true.
- $n_{\text{good}}, n_{\text{bad}}, n_{\text{novec}}$ did not change so **2.** $n_{\text{good}} \leq 3r+2$, **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$, **4.** $n_{\text{novec}} \leq 3r+2$ all remain true.
- n_{rm} did increase but **5.** $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r + 1$ holds since we must have $n_{\text{rm}} \leq r + n_{\text{good}} - 1$ (can't remove more vectors than the total we've added: $r + n_{\text{good}}$, in fact, at least one must remain since $s = \dim(L) \geq \dim(W) \geq 1$) and **2.** $n_{\text{good}} \leq 3r+2$ is true.
- n_{scales} and n_{entries} do not change so **6.** $n_{\text{scales}} \leq 3r+2$ and **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$ remain true.
- **8.** $n_{\text{switches}} \leq P$. It suffices to show **9.** $P^{\text{out}} \geq P^{\text{in}}$ since no switches are made. But **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^{2r}$ and **1.** $s < \lfloor \frac{3r}{2} \rfloor$ so replacing $(s) \cdot l_s$ by $(3r/2) \cdot 2r \log_{\sqrt{4/3}}(2)$ only increases Progress.
- This is the most important place to check **10.** $W \subseteq \pi(L)$.

  Proof. L^{in} satisfies property **11** so for every v_g in the basis of W we know that Fact 1 from 1.3.1 applies. Thus $v_g \in L^{\text{out}}$ for every v_g in the basis of W, which implies **10.** $W \subseteq \pi(L)$. □

- **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r+1$ for any irreducible factor g remains true since $\pi^{-1}(v_g)$ doesn't change during **3.7(Remove Vectors)**. (The entries don't change at all.)
- We now have a B-reduced set, so an appeal to Theorem 8 in section 3.3.2 implies both **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^r$ and **13.** $\text{AD}^{\text{out}} \leq 2^{r^2}$.
- **14.** $\text{AD}^{\text{out}} \geq \frac{1}{2^{2r \cdot \triangle(n_{\text{rm}})}} \cdot \text{AD}^{\text{in}}$ follows from Theorem 4.

3.7.6 Complexity Notes:

- This procedure is called from **3.11(LLL)** (Exit 12) $n_{\text{scales}} + n_{\text{novec}}$ times (see complexity notes in **3.11**).
- This procedure is called from **3.9(pLLL)** (Exits 11a and 11b) $n_{\text{good}} + n_{\text{bad}}$ times (see complexity notes in **3.9**).
- There is only one exit from this procedure (Exit 13) and is thus called $n_{\text{good}} + n_{\text{bad}} + n_{\text{novec}} + n_{\text{scales}}$ times.
- The step can be done very quickly if we modify LLL and pLLL to output not only $b_1, \ldots, b_s$ also their G-S lengths.

3.8 Select Next Coefficient

3.8.1 Variables Used in this Procedure

- N is the degree of f.
- Each loop tests the cth coefficient of the $\frac{f_i' f}{f_i}$ for usability. If the cth coeff is not usable then increase c by one and try again.
- $k_{\max}$ is the index of the current vector who would receive the largest new entry if we were to use the cth coeff. This index is used in **3.12(No Vector Added)** and reassigned in **3.10(Scale Up)**.
- $\psi_c(b_i^{\text{in}})$ is the new entry we would add and equals $\sum_j \text{CLD}_c^a(f_j) \cdot (b_i^{\text{in}})_j$.
- $\text{CB}(c) + d$ is our bounding factor. $2^{\text{CB}(c)} \geq |\text{CLD}_c^a(g)|$ where g is any rational factor of f (see 3.3.2). So if v_g corresponds with a rational factor of f then $\frac{\text{CLD}_c^a(v_g)}{2^{\text{CB}(c)}} \leq 1$. d was decided in **3.4(Initialize)** so that $\frac{1}{2^d} \leq \frac{1}{\sqrt{3r^2+5r+5}}$.
- tc is the current number of coefficients which have been added to the vectors b_i. note: $\text{tc}^{\text{in}} = n_{\text{good}} + n_{\text{bad}} + n_{\text{novec}}$.
- a_{tc} and c_{tc} are book keeping variables so that we can later know the level of p-adic accuracy and which coefficient went in the $(\text{tc} + r)^{\text{th}}$ entry of any vector. These numbers are never changed again.

3.8.2 Procedure

1. For i from 1 to N do:

   (a) If Avail_Bits$(c) \geq 3r$ then:

      i. $k_{\max} := i$ where i any index with $|\psi_c^a(b_i^{\text{in}})| \geq |\psi_c^a(b_j^{\text{in}})| \forall j$

      ii. If $|\psi_c^a(b_{k_{\max}}^{\text{in}})| > 2^{2r} \cdot 2^{\text{CB}(c)+d}$ then:
      (Note: $2^{2r} \cdot 2^{\text{CB}(c)+d} \geq 2^r \cdot \parallel b_{k_{\max}}^{\text{in}} \parallel \cdot 2^{\text{CB}(c)+d}$)

         A. $\text{tc} := \text{tc} + 1$;

         B. $c_{\text{tc}} := c$; Keeps a record of which coefficient was the tc^{th} coefficient we use.

         C. $a_{\text{tc}} := a$; Keeps a record of the p-adic accuracy of the tc^{th} coefficient we use.

         D. Exit this procedure and call **3.12(No Vector Added)** (Exit 8)

      fi;

   fi;

(b) $c := c + 1 \bmod N$;

od;

2. Exit this procedure and call **3.6(Check If Solved)** (Exit 7)

3.8.3 What This Procedure Does:

In the previous chapter we applied Gradual B-Reduction (Algorithm 3) to a special matrix. The reason that the switch complexity had an N in it was because there were $N + r$ columns of the All-Coefficients Matrix from [3] (each of the final N columns being one of the CLD_c^a). **3.7** is a rather simple sub-procedure which is the key to removing the N from the switch complexity, namely, we no longer use every column/coefficient of the logarithmic derivative. The Coefficients we will use have two properties:

- The bound on that particular coefficient is sufficiently lower than our level of p-adic accuracy, so that we can have a vector of length roughly 2^{3r} to add to the lattice.
- At least one new entry is sufficiently large. We formulate this property in a way which will be useful for other proofs.

If no coefficient is left which satisfies these two properties we will go back to **3.5(Hensel Step)** again but first check to see if the problem is solved since a solved problem won't find any usable coefficients either. (Exit 7)

If a usable coefficient is found than we will first check if it is a No Vector Coefficient by calling **3.12(No Vector Added)** (Exit 8).

Finding $n_{\mathbf{hensel}}$ and some Termination Notes

Lemma 10. *There is an n_{hensel} such that if $p^a > p^{n_{hensel}}$ and **3.8** does not succeed (find a usable coefficient and then take Exit 8) then $\pi(L) = W$.*

Begin Proof Assume $\pi(L) \neq W$. Let $f = g_1 \cdots g_k$ be the factorization over $\mathbb{Q}$ of f with degree N, and $f_1, \ldots, f_r$ be the p-adic factors of f. Now define $e_i \in \{0, 1\}$ so that $g_1 = \prod f_i^{e_i}$. Let $w_1 = (e_1, \ldots, e_r, \text{Coeffs of } \frac{g_1' f}{g_1})$. By Cor. 4.2 in [3] we have $\| w_1 \| \leq \sqrt{r + (2^{N-1} \cdot N \cdot \| f \|_2)^2}$. Likewise define $w_2, \ldots, w_k$. Let $B' := \sqrt{r + (2^{N-1} \cdot N \cdot \| f \|_2)^2}$.

1. $\mathrm{CB}(c) \leq \lceil \log_2(2^{N-1} \cdot N \cdot \| f \|_2 \rceil$. See Section 3.3.2 and Cor 4.2 in [3]. Let $M := 2^{(2r + \lceil \log_2(2^{N-1} \cdot N \cdot \| f \|_2 \rceil + d)}$.

2. Procedure **3.8** succeeds (uses Exit 8) if $\exists i$ with $|\psi_c^a(b_i)| > 2^{(2r+\text{CB}(c)+d)}$.

3. $W \subseteq \pi(L)$ so if $\pi(L) \neq W$ then $\exists i$ with $\pi(b_i) \notin W$.

4. $\| b_i \| \leq 2^r$ (see Lemma 8 in Section 3.3.3).

5. Let $v = \pi(b_i)$ from 3. Let $v = (v_1, \ldots, v_r)$, and let $\tilde{v} = (v_1, \ldots, v_r, \psi_0(v), \ldots \psi_{N-1}(v))$ mods p^a.

6. If $\| \tilde{v} \|^2 > (r+N) \cdot M^2$ then **3.8** succeeds, by 1, 2, and 4. Let $K := \sqrt{r+N} \cdot M$.

7. Now we follow the proof of Theorem 4.3 in [3]. We only mention the parts we need for our computation:

   First adjust $\tilde{v}$ so that it will satisfy the conditions in Lemma 3.2 from [3], which produces: $b = \tilde{v} + \sum_{j=1}^{k} n_j w_j$. Here $n_j \in \{0,1\}$ for all but one j and $n_j \in \{0-e, 1-e\}$ for one j, where $e \in \{v_1, \ldots, v_r\}$, see [3] for details. So $|e| \leq 2^r$ by 4.

   Thus if $\tilde{v} \leq K$ then $\| b \| \leq K + (|e| + k) \cdot B' \leq K + (2^r + r) \cdot B'$ which define as B''.

   Let the entries of $b = (b_1, \ldots, b_r, u_0, \ldots, u_{N-1})$. Let $H := \sum u_i x^i$.

   Then $0 < \text{Res}(H, f) \leq (B'' \cdot \| f \|_2)^N$, and $p^a | \text{Res}(H, f)$.

   Therefore $p^a \leq (B'' \| f \|_2)^N = 2^{\mathcal{O}(N^2 + N \log(\|f\|_2))}$.

   Now define $n_{\text{hensel}} := \log_p((B'' \cdot \| f \|_2)^N)$.

8. Hence if $a > n_{\text{hensel}}$ and $\pi(L) \neq W$ then procedure **3.8** succeeds.

End Proof

3.8.4 Properties true of the Input

- To Confirm these properties check **3.10(Scale Up)** (Exit 16) and **3.5(Hensel Step)** (Exit 5).
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$

3.8.5 Properties true of Exit 8

The current coefficient, c, might have changed and it was chosen in step 1a so that there is at least 2^{3r} available information (between our coefficient bound and the accuracy of our hensel lifting so far), and at least one entry has usable size larger than 2^r times the vector without the new entry (see step 1(a)ii and the fact that $\| b^{\text{in}}_{k_{\max}} \| \leq 2^r$ (property 12in)).

No other variables changed and no vectors changed so the input properties holds still.

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^r$
- **13.** $\text{AD}^{\text{out}} \leq 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$

  Also:

- Avail_Bits$(c) \geq 3r$
- There is a large entry that we can add:
  $|\psi_c^a(b^{\text{in}}_{k_{\max}})| > 2^{2r} \cdot 2^{\text{CB}(c)+d} \geq 2^r \cdot \| b^{\text{in}}_{k_{\max}} \| \cdot 2^{\text{CB}(c)+d}$

3.8.6 Properties true of Exit 7

No variables except c changed and no vectors changed so

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^r$
- **13.** $\text{AD}^{\text{out}} \leq 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$

all remain true.

3.8.7 Complexity Notes:

- This procedure is called from **3.5(Hensel Step)** (Exit 5) at most $\log_2(n_{\text{hensel}}) - \log_2(\text{IB})$ times. (See **3.5**.)
- This procedure is called from **3.10(Scale Up)** (Exit 16) at most $n_{\text{novec}} + n_{\text{good}} + n_{\text{bad}}$ times. (See **3.10**.)
- There are only two procedures which call this procedure, so the total number of times this procedure is called is $\leq n_{\text{novec}} + n_{\text{good}} + n_{\text{bad}} + \log_2(n_{\text{hensel}}) - \log_2(\text{IB})$ times.
- This procedure calls **3.12(No Vector Added)** (Exit 8) every time there is a usable coefficient. Every usable coefficient will increase the counter n_{novec} or n_{good} or n_{bad} so this exit is used $n_{\text{novec}}+n_{\text{good}}+n_{\text{bad}}$ times.
- There are only two exits, so this procedure can only call **3.6(Check If Solved)** (Exit 7) $\leq \log_2(n_{\text{hensel}}) - \log_2(\text{IB})$ times.
- Each loop might check N coefficients, which creates a complexity term with N in it, but it is not related to switch complexity.

3.9 pLLL the Probationary LLL run

3.9.1 Variables used in this Procedure

- i is just a local variable for looping through the b_i.
- c is the current coefficient of $\frac{g'f}{g}$ which will add to our vectors. $\psi_c^a(b_i)$ is congruent mod p^a to the cth coefficient of $\frac{g'f}{g}$, where $g = \prod_{j=1}^{r} f_j^{(b_i)_j}$.
- ISD is the initial scale down, which was[1] chosen in **3.12(No Vector Added)**. This was chosen so that $2^{3r-1} < \frac{p^a}{2^{\text{ISD}}} \leq 2^{3r}$. This is the first scaling done to a new coefficient and is done to all new entries added in this step. This first scale down corresponds roughly to step 1 in Gradual B-Reduce.
- n_{entries} is increased by one during this procedure so we will need to prove that **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$ remains true. n_{entries} was defined first during **3.4(Initialize)**, and only increases here and in **3.12**.

[1] The word ‘was’ refers to the fact the **3.12** comes before **3.9** in the algorithm. In the thesis we present **3.9** first because these concepts are central to the rest of the thesis.

- n_{switches} is increased during this procedure once for every LLL switch which takes place, and only changes during this procedure and **3.11(LLL)**.

- Either n_{bad} or n_{good} will increase in this procedure and this is the only procedure where these variables change.

- s is the number of vectors in the active basis of L, and $\| b_s^* \|$ is the G-S length of the final vector. This procedure and **3.7(Remove Vectors)** are the only procedures where s might change. In this procedure s increases by one if we are in the Good Coeff Case and stays the same if we are in the Bad Coeff Case.

3.9.2 Procedure

1. For i from $s+1$ to 2 do

   $b_i := (b_{i-1}, \frac{\psi_c^a(b_{i-1})}{2^{\text{ISD}}})$; (Appended Entries and shifted indices)

   od;

2. $b_1 := (\vec{0}, \frac{p^a}{2^{\text{ISD}}})$;

3. $s := s+1$ (We now have one more vector and have increased AD by a factor $\frac{p^a}{2^{ISD}}$.)

4. $n_{\text{entries}} := n_{\text{entries}} + 1$;

5. Run LLL on $(b_1, \ldots, b_s)$ and update n_{switches} .

6. **Bad Coeff Case:** If $\| b_s^* \| > 2^{2r}$ then

   (a) $n_{\text{bad}} := n_{\text{bad}} + 1$ and

   (b) Remove b_s (don't increase n_{rm})

   (c) $s := s - 1$

   (d) Exit this procedure and call **3.7(Remove Vectors)** (Exit $11b$)

   fi;

7. **Good Coeff Case:** If $\| b_s^* \| \leq 2^{2r}$ then

   (a) $n_{\text{good}} := n_{\text{good}} + 1$

   (b) Exit this procedure and call **3.7(Remove Vectors)** (Exit $11a$)

fi;

3.9.3 What This Procedure Does:

This procedure adds a new entry and new vector in the same manner as algorithm 4 in Section 2.2 did, so n_{entries} and s are both increased by 1. This is the first time LLL is called on the new entry, and this is the only time in the entire algorithm which allows a vector to have G-S length $> 2^{2r}$. We use the term probationary:

Definition 9. *Any vector with G-S length $> 2^{2r}$ is called a probationary vector.*

The vector $b_1 = (\vec{0}, \frac{p^a}{2^{\text{ISD}}})$ in step 2 is probationary when step 5 is called, and it plays several important roles in this algorithm:

- When step 2 makes this vector the first vector in our basis, b_1, we can see that the G-S lengths of $b_2, \ldots, b_s + 1$ are unaltered by their new entry (i.e. $\| b_i^* \| = \| (b_{i-1}^{\text{in}})^* \|$).
- Since L is the span of the b_i the introduction of this vector, b_1, means that any v in L^{in} has a corresponding vector, $\tilde{v}$ for which the newly added last entry is reduced mods p^a (by adding or subtracting b_1).
- Because of our choice for ISD we know that $\| b_1^* \| = \| b_1 \| > 2^{3r-1}$, which means AD increased by at least a factor 2^{3r-1}.

Recall that we need the G-S length of any added vectors to be larger than the G-S length of any removed vectors. This was in order to make sure that the algorithm eventually removes more vectors than it adds. The **3.9** procedure is what ensures that the length of the added vectors is at least 2^{3r-1}.

Theorem 5. *Every time n_{good} is increased by 1 we must have AD multiplied by a factor $\geq 2^{3r-1}$, so that it is always true $AD \geq 2^{(3r-1)n_{good} + (-2r)n_{rm}}$.*

Proof. We add a vector of size $\geq 2^{3r-1}$ while the other G-S lengths stayed fixed which proves the first part. The second follows from Theorem 4 in **3.7**. □

While generally the G-S length of a removed vector is $\leq 2^{2r}$. The one possible exception is during the first time LLL is called with the new entry, in which case a vector of size larger than 2^{2r} can survive LLL. We call this the Bad Vector Case because the large removed vector leads to a worse bound for how many times a new vector can be added. Was it possible that some second large vector became probationary? No because we carefully choose 2^{2r} as the cutoff for probationary status:

Lemma 11. *There exists at most one probationary vector.*

Proof. We know that $\| (b_i^{\text{in}})^* \| \leq 2^r$, $\| b_1^* \| = \frac{p^a}{2^{\text{ISD}}} \leq 2^{3r}$, and $\| b_j^* \| = \| (b_{j-1}^{\text{in}})^* \| \leq 2^r$ just before step 5.

During this step LLL switches are made which may involve a probationary vector and may not. When LLL switches two vectors, b_i, b_{i+1} there are some properties we must rely on: the product of their G-S lengths is fixed and the $\max(\| b_i^* \|, \| b_{i+1}^* \|)$ cannot increase from the switch. So every time the second largest G-S length (in the entire basis) increases the largest G-S length must have decreased. However when the problem began the largest G-S length was 2^{3r} and the second largest 2^r. So if a switch involves both the largest and second largest vector then 2^{3r+r} must be the product of their G-S lengths after and before the switch. Suppose the largest is now $2^{3r-\alpha}$ then the G-S length of the formerly second largest is $2^{r+\alpha}$. But if $\alpha \neq r$ then one vector has G-S length $> 2^{2r}$ and one below. If $\alpha = r$ then they both have G-S length $= 2^{2r}$. □

Lemma 12. *If there exists a probationary vector after step 5 it must be the last vector.*

Proof. The G-S lengths of $b_2, \ldots, b_{s+1}$ are actually all $\leq 2^{r-1}$ (Theorem 8 in section 3.3.3) before step 5. Let's say that there are exactly k non-probationary vectors of G-S length $2^{r-1+\alpha_i} > 2^{r-1}$ after the LLL run in step 5, and let's say that the probationary vector is of length $2^{2r+\epsilon}$. We know that $\alpha_1 + \cdots + \alpha_k \leq r - \epsilon$ because any G-S length above 2^{r-1} must have been taken away from the largest vector (or passed it around to other vectors).

However after LLL we know that this set is LLL reduced so $\| b_i^* \|^2 \leq 2^j \| b_{i+j} \|^2$ for $j \geq 0$. Now suppose that the probationary vector is the $(s-j)^{\text{th}}$ vector, and let $\| b_{s-j+i}^* \| = r - 1 + \alpha_i$ for i from 1 to j. Then $4r + 2\epsilon \leq 2r - 2 + 2\alpha_i + i$ so $\alpha_i \geq r + 1 + \epsilon - i/2$. However we saw before that $\sum \alpha_i \leq r - \epsilon$. This implies that $jr + j + j\epsilon - (1/2)(j(j+1)/2) \leq r - \epsilon$. This equation is true when $j = 0$, but not when $j = 1$ and the left hand side of this equation increases with j for $j \leq 2r + 2\epsilon - 1/2$. But $j \leq s$ which is $\leq 3r/2 + 1$ by **1.** $s < \lfloor \frac{3r}{2} \rfloor$. (+1 just to be safe since we've yet to prove property 1 for the output but s only increases by 1). □

Notation: If there still remains a probationary vector after running LLL in step 5 then we call the probationary vector a **Bad Vector**. We choose the word bad because in this case AD^{in} might be only marginally more than AD^{out}, making the bound for n_{bad} quadratic in r not linear.

3.9.4 Properties true of the Input

- To Confirm these properties check **3.12** Exit 9.
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)

- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$
- $\text{Avail_Bits}(c) \geq 3r$
- There is a large potential entry:
  $|\psi_c^a(b_{k_{\max}}^{\text{in}})| > 2^{2r} \cdot 2^{\text{CB}(c)+d} \geq 2^r \cdot \| b_{k_{\max}}^{\text{in}} \| \cdot 2^{\text{CB}(c)+d}$

3.9.5 Properties of BOTH Exits $11a$ and $11b$

Properties **8** and **11** have long proofs which are the same in both the good and the bad case so we will include them in this separate section. We also prove **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$ and **9.** $P^{\text{out}} \geq P^{\text{in}}$.

- Now proving **8.** $n_{\text{switches}} \leq P$, requires showing that each switch increased Progress by at least 1 but this is the same in both good and bad case:

  Statement 1. *When LLL makes a switch it involves consecutive vectors, let them be b_i and b_{i+1}, let l_i and l_{i+1} be the log of their G-S lengths. Let b_i', b_{i+1}', l_i', and l_{i+1}' be the vectors and their logarithmic G-S lengths after the switch. Then $l_i \geq l_{i+1}$, $l_{i+1} \leq l_i' \leq l_i - 1$, $l_{i+1} + 1 \leq l_{i+1}' \leq l_i$, and $l_i + l_{i+1} = l_i' + l_{i+1}'$. Also the G-S lengths of each of the other vectors remain unaltered by the switch.*

  Statement 2. *Switches not involving the probationary vector in step 5 increase progress by at least 1.*

  Proof. The vectors that this statement considers are weighted by il_i in P. Using the notations and facts from statement 1 we can compare progress contributed from the two vectors in question before and after the switch. Call progress contributed from two vectors before the switch $P_b := il_i + (i+1)l_{i+1}$ and after: $P_a := il_i' + (i+1)l_{i+1}'$. Observe: $P_b + 1 = il_i + (i+1)l_{i+1} + 1 = i(l_i' + l_{i+1}' - l_{i+1}) + (i+1)l_{i+1} + 1 = il_i' + il_{i+1}' + (l_{i+1} + 1) \leq il_i' + (i)l_{i+1}' + (l_{i+1}') = il_i' + (i+1)l_{i+1}' = P_a$. □

  Statement 3. *Switches involving the probationary vector increase progress by at least 1.*

  Proof. The contribution of the probationary vector, b_i, to progress, P, is just $i - 1$, and the other vectors are weighted as if the probationary vector doesn't alter their index. So the weight of any vector, b_j with $j > i$ is $(j-2)l_j$ and with $j < i$ is $(j-1)l_j$.

  When making a switch involving the probationary vector it must be the vector with lower index since it is the vector with largest l_i. So the Progress contributed by the two involved vectors before the switch

will be denoted $P_b = (i-1) + il_{i+1}$. There are three outcomes from the switch, the probationary vector keeps the same index, increases its index, or becomes a good vector (G-S length below 2^{2r}). Each one has a different contribution to P, which we denote P_a:

Case 1: Probationary vector keeps same index: $P_a := (i-1) + il'_{i+1}$. So $P_b + 1 = (i-1) + il_{i+1} + 1 \leq (i-1) + i(l_{i+1} + 1) \leq (i-1) + i(l'_{i+1}) = P_a$. (This required $i \geq 1$.)

Case 2: Probationary vector increases its index: $P_a := il'_i + (i)$. So $v_b + 1 = (i-1) + i(l_{i+1}) + 1 \leq i(l'_i) + (i) = P_a$.

Case 3: Probationary vector now has G-S length $\leq 2^{2r}$: $P_a := il'_i + (i+1)l'_{i+1}$ (and in this case alone all of the vectors of index $> i+1$ increase their weight to jl_j but $l_i \geq 0$ so this effect cannot decrease progress P). So if $l_{i+1} \geq 0$ and if $l'_i \geq 0$ then $P_b + 1 \leq i + (i+1)l_{i+1}$ $\leq [i + (i+1)l_{i+1}] + 1 + il'_i = (i+1)(l_{i+1} + 1) + il'_i \leq il'_i + (i+1)(l'_{i+1}) = P_a$. □

- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$ follow from construction in the good case. In the bad case we have removed one vector of size $> 2^{2r}$ but lemma 11 ensures us that only one such vector existed.

- **9.** $P^{\text{out}} \geq P^{\text{in}}$ requires the above proofs for **8.** $n_{\text{switches}} \leq P$, and the fact that the removal of a probationary vector can only increase P. Since it is only weighted by its index and **1.** $s < \lfloor \frac{3r}{2} \rfloor$ ensures us that $(s-1) \leq 3r/2$.

- **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r + 1$ for any irreducible factor g.

Proof. We've increased the number of entries so all that we need to check is if the new entry has size $\leq \frac{1}{\sqrt{3r^2+5r+5}}$.

But since $(\pi^{-1}(v_g))^{\text{in}} \in L^{\text{in}}$ and we add a new entry which is $\frac{\psi_c^a(v_g)}{2^{\text{ISD}}}$ to $(\pi^{-1}(v_g))^{\text{in}}$. But the existence of the vector $(\vec{0}, \frac{p^a}{2^{\text{ISD}}})$ in L allows us to reduce the last entry mod p^a. So $(\pi^{-1}(v_g))^{\text{out}}$ has last entry $\frac{\text{Coeff}_c^a(v_g)}{2^{\text{ISD}}}$. So an appeal to Fact 2 in 3.3.3, shows a sufficiently small last entry.

Note that when we are in the bad vector case we removed a vector of G-S length $\geq 2^{2r} > B$, so $\pi^{-1}(v_g) \in L^{\text{out}}$ by Fact 1 in section 1.3.1.

□

3.9.6 Properties true at Exit $11b$

- $s, n_{\text{good}}, n_{\text{novec}}, n_{\text{rm}}, n_{\text{scales}}$ didn't change so **1.** $s < \lfloor \frac{3r}{2} \rfloor$, **2.** $n_{\text{good}} \le 3r+2$, **4.** $n_{\text{novec}} \le 3r+2$, **5.** $n_{\text{rm}} \le r + n_{\text{good}} - 1 \le 4r+1$, **6.** $n_{\text{scales}} \le 3r+2$ all remain true.

- **10.** $W \subseteq \pi(L)$ follows from Fact 1 in section 1.3.1 when the one removed vector is the final vector.

- **13.** $\text{AD}^{\text{out}} \le 2^{r^2+3r-2r}$ since adding the probationary vector increased the AD by no more than 2^{3r} and ensured that the other G-S lengths were unaffected and the removed vector had G-S length $> 2^{2r}$.

- **14.** $\text{AD}^{\text{out}} \ge 2 \cdot \text{AD}^{\text{in}}$ requires a rather long proof:

Statement 4. *If the outcome of **3.9** is a bad vector (n_{bad}) then the net effect of **3.9** on AD is multiplication by a factor of at least 2.*

Proof. $S := B \cdot (2(3/2)^{(s-1)} - 2)$, $T := \frac{p^a - 2^{\text{ISD}} \cdot B}{K}$, and $K := |\psi_c^a(b_{k_{\max}})|$ come from **3.12**.

We use the following facts: $B \le 2^{r/2}$, $s \le 3r/2$, $S \ge T$, and $S < 2^s - 1$.

Let $w_0 := (\vec{0}, \frac{p^a}{2^{\text{ISD}}})$ and $M := \| w_0 \| = \frac{p^a}{2^{\text{ISD}}}$.

CLAIM: If $S \ge T$ then there is at least one k with $|(b_k^{\mathbf{out}})_{-1}| > 2 \cdot |b_k^{\mathbf{in}}|$

Suppose not, then $|\psi_c^a(b_i^{\text{in}})| \le 2^{\text{ISD}} \cdot 2 \cdot \| b_i^{\text{in}} \| \le 2^{r+1+\text{ISD}}$ for all i, so $K \le 2^{r+1+\text{ISD}}$. (Since **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \le 2^r$.)

Note that $S < 2^s - 1$, that $B \le 2^{r/2}$, and that $\frac{p^a}{2^{3r-1}} > 2^{\text{ISD}} \ge \frac{p^a}{2^{3r}}$.

So $S \cdot K + B \cdot 2^{\text{ISD}} \le (2^s - 1)(2^{r+1})(\frac{p^a}{2^{3r-1}}) + (2^{r/2})(\frac{p^a}{2^{3r-1}}) = p^a[2^{s+r+1-3r+1} - 2^{r+1-3r+1} + 2^{r/2-3r+1}]$ and we know **1.** $s < \lfloor \frac{3r}{2} \rfloor$ and $r \ge 10$ so this is $\le p^a[\frac{4}{2^{r/2}} - \frac{4}{2^{2r}} + \frac{2}{2^{5r/2}}] \le p^a$.

But then $S < T$ a contradiction.

End of Claim's Proof.

Let the LLL reduced set returned by LLL in step 5 be denoted by $v_1, \ldots, v_s, v_0$ and we will define vectors $c_1, \ldots, c_s$ so that $v_i = (c_i, (v_i)_{-1})$. We also split up v_0 as $(\vec{v}, \alpha)$.

For ease we will call $w_0 = (\vec{0}, M)$.

Now let $L' := \text{SPAN}_{\mathbb{Z}}(c_1, \ldots, c_s)$ and note that $c_i \in L^{\text{in}}$ for all i so $L' \subseteq L^{\text{in}}$.

If $\| v_0^* \| \le M/2$ then $\prod_{i=1}^{s} \| v_i^* \| \ge 2 \cdot \text{AD}^{\text{in}}$ since $\text{AD}^{\text{in}} \cdot M = \prod_{i=0}^{s} \| w_i^* \|$ which $= \| v_0^* \| \cdot \prod_{i=1}^{s} \| v_i^* \|$ which is $\| v_0^* \| \cdot \text{AD}^{\text{out}}$.

So if the following is true then we are done: **To Prove:** $\| v_0^* \| \leq M/2$ So suppose that $\| v_0^* \| > M/2$ then there are two cases to be treated:

Case 1: $L' \neq L^{\text{in}}$.

Then $\det(L') > \det(L^{\text{in}})$ but $[L^{\text{in}} : L'] \in \mathbb{Z}$ so $\det(L') \geq 2\det(L^{\text{in}})$.

However $\prod_{i=1}^{s} \| v_i^* \| \geq \det(L') \geq 2\det(L^{\text{in}}) = 2 \cdot \text{AD}^{\text{in}}$ which would give us the outcome we desire which indirectly contradicts the assumption.

Case 2: $L' = L^{\text{in}}$.

Then c_i are linearly independent so $(\vec{0}, M)$ cannot be in $\text{SPAN}_{\mathbb{Z}}(v_1, \ldots, v_s)$, while it is in $\text{SPAN}_{\mathbb{Z}}(v_1, \ldots, v_s, v_0)$. If we also observe that $v \in L^{\text{in}} = L'$ we reveal that there must be $a_i \in \mathbb{Z}$ with $v_0 = (-1)^n(\vec{0}, M) + \sum_{i=1}^{s}(a_i) \cdot v_i$.

Also if k is the index ensured by the above claim then $w_k \in \text{SPAN}_{\mathbb{Z}}(v_1, \ldots, v_s)$ since $b_k^{\text{in}} \in L^{\text{in}} = L'$.

All of this leads to the fact that there must be real numbers $r_1, \ldots, r_s$ such that $v_0 - \sum_1^s (r_i v_i) = (-1)^n(\vec{0}, M) + (-1)^{n+1}(\frac{M}{(w_k)_{-1}}) \cdot w_k$ which has length $\leq M/2$.

But this bounds the G-S length of v_0 and we prove the contradiction.

Thus in either case the Bad Vector Case multiplied AD by a factor ≥ 2. $\square$

It remains to show: **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$ and **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$.

- To prove **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$ we want to show that:

  Lemma 13. $n_{bad}^{in} \leq 3r^2 - 2r$

  Proof. To bound n_{bad} we recall that $2^{r^2} \geq \text{AD}^{\text{in}}$. But since n_{bad} is a counter and every time it increases by one we know that AD increases by a factor 2, and we know that Theorem 4 in **3.7** ensures us that AD is only ever decreased by increasing n_{rm} we must have $\text{AD}^{\text{in}} \geq 2^{n_{\text{bad}}^{\text{in}} + (-2r)n_{\text{rm}}}$. Also $n_{\text{rm}} \leq r + n_{\text{good}} - 1$ and every time n_{good} increases AD increased by more than $2^{3r-1} > 2^{2r}$ (see Theorem 5 in **3.9**) so to get an upper bound for n_{bad} we should use $n_{\text{rm}} = r - 1$. So we have $2^{r^2} \geq 2^{n_{\text{bad}}^{\text{in}} - 2r^2 + 2r}$ and thus $n_{\text{bad}}^{\text{in}} \leq 3r^2 - 2r$. $\square$

  Now just observe that $n_{\text{bad}}^{\text{out}} = n_{\text{bad}}^{\text{in}} + 1$.

- **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$ now follows from **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$ with **2.** $n_{\text{good}} \leq 3r + 2$, **4.** $n_{\text{novec}} \leq 3r + 2$ and fact that $n_{\text{entries}} = r + n_{\text{good}} + n_{\text{bad}} + n_{\text{novec}}$.

3.9.7 Properties true at Exit $11a$

- $n_{\text{bad}}, n_{\text{novec}}, n_{\text{rm}}, n_{\text{scales}}$ don't change so **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$, **4.** $n_{\text{novec}} \leq 3r + 2$, **5.** $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r + 1$, and **6.** $n_{\text{scales}} \leq 3r + 2$ all remain true.

- **10.** $W \subseteq \pi(L)$ is true because $\pi(L)$ is invariant under new entries.

- **13.** $\text{AD}^{\text{out}} \leq 2^{r^2+3r}$ since we added a new vector of G-S length $\leq 2^{3r}$, the other G-S lengths were unaltered, and LLL switches preserve AD.

- **14.** $\text{AD}^{\text{out}} \geq 2^{3r-1} \cdot \text{AD}^{\text{in}}$ by the same reasoning (see Step 3).

  Remains to show: **1.** $s < \lfloor \frac{3r}{2} \rfloor$, **2.** $n_{\text{good}} \leq 3r + 2$, and **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$:

- **1.** $s < \lfloor \frac{3r}{2} \rfloor$ requires some extra thought in this section since s might have increased by 1, so we will prove $s^{\text{in}} < \lfloor 3r/2 \rfloor - 1$:

  Statement 5. $AD^{in} \leq 2^{(s-1)s/4+(sr/4)}$

  Proof. Since we began with a B-reduced set and $B = r + 1 < 2^{r/2}$ for $r > 10$ we know that $\| b_s^* \| \leq 2^{r/4}$. Also the definition of LLL reduced implies that $\| b_{s-i}^* \| \leq 2^{r/4+i/2}$. Thus $\text{AD} = \prod \| b_i^* \| \leq 2^{sr/4+1/2\sum_0^{s-1} i} \leq 2^{sr/4+(s-1)(s/4)}$. □

  Statement 6. $AD^{in} \geq 2^{(3r-1)(s-r)}$

  Proof. Every time s has increased AD has been multiplied by $> 2^{3r-1}$ and when s has decreased AD has been divided by something $< 2^{3r-1}$. Also our starting basis had r vectors. □

  Statement 7. $s^{in} < \lfloor 3r/2 \rfloor - 1$

  Proof. $s^{\text{in}} \leq \lfloor 3r/2 \rfloor - 1$ by property 1. It just remains to prove that $s^{\text{in}} \neq \lfloor 3r/2 \rfloor - 1$. First note that: $(3r/2 - 1) \geq \lfloor 3r/2 \rfloor - 1 > 3r/2 - 2$. If we can show that $2^{(3r-1)(s-r)} > 2^{(s-1+r)(s/4)}$ for $s = \lfloor 3r/2 \rfloor - 1$ then we will arrive at a contradiction between the previous two statements. So using the above inequality we see that $(3r-1)(\lfloor 3r/2 \rfloor - 1 - r) > (3r-1)(3r/2 - 2 - r) = 3r^2/2 - 13r/2 + 2 = Q_1$ and $(s/4)(s - 1 + r) \leq (3r/8 - 1/4)(3r/2 - 2 + r) = 15r^2/16 - 11r/8 + 1/2 = Q_2$.

  If we can show that $Q_1 - Q_2 > 0$ for $r > 10$ then we are done. But $Q_1 - Q_2 = 9r^2/16 - 43r/8 + 3/2$ which is increasing for $r > 5$ and $Q_1 - Q_2 = 4$ when $r = 10$. □

- **2.** $n_{\text{good}} \leq 3r + 2$ requires the following lemma:

Lemma 14. $n_{good}^{in} \leq 3r + 1$

Proof. We know that $2^{r^2} \geq \mathrm{AD}^{\mathrm{in}}$. Also theorem 4 in section **3.7** ensures us that only n_{rm} decreases AD. Every time n_{good} increased so does AD by a factor 2^{3r-1}, so $\mathrm{AD}^{\mathrm{in}} \geq 2^{(3r-1)n_{\mathrm{good}}^{\mathrm{in}} - 2rn_{\mathrm{rm}}}$. But $n_{\mathrm{rm}} \leq n_{\mathrm{good}}^{\mathrm{in}} + r - 1$, so $\mathrm{AD}^{\mathrm{in}} \geq 2^{(r-1)n_{\mathrm{good}}^{\mathrm{in}} - 2r^2 + 2r}$. Altogether we get $2^{3r^2-2r} \geq 2^{(r-1)n_{\mathrm{good}}^{\mathrm{in}}}$, which implies that $n_{\mathrm{good}}^{\mathrm{in}} \leq 3r + 1$. □

- **7.** $n_{\mathrm{entries}} \leq 3r^2 + 5r + 5$ now follows from **2.** $n_{\mathrm{good}} \leq 3r + 2$, **3.** $n_{\mathrm{bad}} \leq 3r^2 - 2r + 1$, **4.** $n_{\mathrm{novec}} \leq 3r + 2$, and fact that $n_{\mathrm{entries}} = r + n_{\mathrm{good}} + n_{\mathrm{bad}} + n_{\mathrm{novec}}$

3.9.8 Complexity Notes:

- This procedure is only called from **3.12(No Vector Added)** (Exit 9), and every time it is called either n_{good} or n_{bad} is increased so the number of calls to **3.9(pLLL)** is $n_{\mathrm{good}} + n_{\mathrm{bad}}$.

- Every time we leave the procedure in the Bad Coeff Case n_{bad} is increased so Exit 11b happens n_{bad} times.

- Every time we leave the procedure in the Good Coeff Case n_{good} is increased so Exit 11a happens n_{good} times.

- The total complexity of this step is split into the LLL costs, and the non-LLL costs. The non-LLL costs are very small compared with the algorithm's overall complexity, and the LLL costs are going to be lumped with the costs of **3.11(LLL)**.

3.10 Scale Up

3.10.1 Variables used in this Procedure

- The notation $(b_j)_{-1}$ is used to represent the final entry of the vector b_j.

- $k_{\max}$ is local to this procedure (the same notation is used in **3.8(Find Next Coeff)** and **3.12(No Vector Added)**). It is simply the index of the basis vector with maximal final entry.

- MSU and CSU are global variables which are defined in **3.12**, and updated in this procedure. MSU stands for the Maximum Scale Up, and is chosen so that we never scale beyond MSU in order to help Property 11stay true throughout the algorithm. CSU, stands for Current Scale Up, and is the running count of how much scaling up has already taken place on the last entry. Note that Scaling Up is distinguished from the Scaling Down done in **3.12** and **3.9(pLLL)**.

- ISD and ESD are the scaling down terms from **3.12**. They are only included here to emphasize that $\mathrm{TS}_{\mathrm{tc}}$ is always $\mathrm{ISD} + \mathrm{ESD} - \mathrm{CSU}$.

- $\mathrm{TS}_{\mathrm{tc}}$ is the term for Total Scaling done to the final entry of our vectors. This is done so that we know that $(\pi^{-1}(v_g))_{-1}$ is always $\frac{\mathrm{CLD}_c^a(v_g)}{2^{\mathrm{TS}_{\mathrm{tc}}}}$. Think of it as storing the amount of scaling done to the $(r+\mathrm{tc})^{\mathrm{th}}$ entry of any vector in L.

- l is also local to this procedure and is a chosen so that

$$2^{2r-1-\epsilon} < 2^l \cdot |(b_{k_{\max}})_{-1}| \leq 2^{2r-\epsilon}$$

  and $\mathrm{CSU} + l \leq \mathrm{MSU}$. The first condition is so that enough information is added by this scaling to justify calling **3.11(LLL)**. The second condition is to make sure we don't scale up by more than we the original scale down. If there is no such l then this coefficient must be milked dry (the final entries are all fairly small).

- n_{scales} is the counter for how many successful scalings have taken place so far, and is only changed in this procedure. In the worst-case complexity estimate $n_{\mathrm{scales}} = 0$ since every successful scaling contributes as much information as a Good Coefficient.

3.10.2 Procedure

1. Find and define $k_{\max}$ an index with $|(b_{k_{\max}})_{-1}| \geq |(b_j)_{-1}| \forall j$

2. If there is an $l \in \{0, \ldots, (\mathrm{MSU} - \mathrm{CSU})\}$ with $2^{2r-1-\epsilon} < 2^l \cdot |(b_{k_{\max}})_{-1}| \leq 2^{2r-\epsilon}$ then

   (a) $n_{\mathrm{scales}} := n_{\mathrm{scales}} + 1$; (Successful scaling)

   (b) $\mathrm{CSU} := \mathrm{CSU} + l$; (Current Scale Up)

   (c) $\mathrm{TS}_{\mathrm{tc}} := \mathrm{ISD} + \mathrm{ESD} - \mathrm{CSU}$;

   (d) for i to s do

   $(b_i)_{-1} := (b_i)_{-1} \cdot 2^l$

   od;

   (e) Exit this procedure and call **3.11(LLL)** (Exit 15)

   fi;

3. c:=c+1;

4. exit this procedure and call **3.8(Find Next Coeff)** (Exit 16)

3.10.3 What This Procedure Does:

This procedure is made to squeeze every bit of data out of our current coefficient before moving on to a new coefficient ($c \to c+1$).

This is very similar to the scaling which takes place in Gradual B-Reduce from the previous chapter (Step 3b of Algorithm 3 in Section 2.1). The difference is that rather than scaling by a fixed amount we actually scale until we have at least one vector with G-S length $> 2^{2r-1-\epsilon}$ and no more than it takes to have all vectors with G-S length $\leq 2^{2r-\epsilon}$. This algorithm will also never allow the current coefficient to be scaled by more than MSU the maximum scale up. If the last entries are still too small after the maximum scale up then we move on to the next coefficient (Exit 16).

If we found a successful scaling then we increase n_{scales} continue to track CSU and TS (the Total Scaling so far on this entry), scale each last entry by 2^l, and call **3.11(LLL)** satisfied that progress will be made. This is Exit 15.

3.10.4 Properties true of the Input

- To confirm these properties check **3.6(Check If Solved)** (Exit 14).
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$

3.10.5 Properties true at Exit 15

- $s, n_{\text{good}}, n_{\text{bad}}, n_{\text{novec}}, n_{\text{rm}}, n_{\text{entries}}$ have not changed so **1.** $s < \lfloor \frac{3r}{2} \rfloor$, **2.** $n_{\text{good}} \leq 3r+2$, **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$, **4.** $n_{\text{novec}} \leq 3r+2$, **5.** $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r+1$, and **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$ still remain true.
- **8.** $n_{\text{switches}} \leq P$ and **9.** $P^{\text{out}} \geq P^{\text{in}}$ are equivalent and follow from fact 1 in Section 2.1 which states that multiplying an entry by a positive scalar can only increase G-S lengths.
- **10.** $W \subseteq \pi(L)$ remains true since $\pi(L)$ is unaffected by scaling one of the later entries.

- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$

Proof. We are given **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$ and that $b_1^{\text{in}}, \ldots, b_s^{\text{in}}$ is a B-reduced (defined in Section 2.1) set. So in particular we know that $\| b_i \| \leq 2^r$ (actual lengths).

There is a new last entry for each b_i, $|(b_i^{\text{out}})_{-1}|$, and we chose l so that $|(b_i^{\text{out}})_{-1}| \leq 2^{2r-\epsilon}$. So $\| b_i^{\text{out}} \| = \sqrt{(\| b_i^{\text{in}} \|)^2 + |(b_i^{\text{out}})_{-1}|^2}$

Which is $\leq \sqrt{2^{2r} + 2^{4r-2\epsilon}} = 2^{2r}(\sqrt{\frac{1}{2^{2r}} + \frac{1}{2^{2\epsilon}}}) \leq 2^{2r}$.

These are actual lengths which implies that $\| b_i^* \| \leq 2^{2r}$.

□

- **13.** $\text{AD}^{\text{out}} \leq 2^{3r^2}$ by **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$ and **1.** $s < \lfloor \frac{3r}{2} \rfloor$. (This is an unimportant bound.)

- **14.** $\text{AD}^{\text{out}} \geq 2^{r-1-\epsilon}\text{AD}^{\text{in}}$

Proof. We are given **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$ and that $b_1^{\text{in}}, \ldots, b_s^{\text{in}}$ is a B-reduced (defined in Section 2.1) set. So in particular we know that $\| b_i \| \leq 2^r$ (actual lengths) (see Theorem 8 in 3.3.3).

There is a new last entry for each b_i, $|(b_i^{\text{out}})_{-1}|$, and we chose l so that $|(b_i^{\text{out}})_{-1}| \leq 2^{2r-\epsilon}$. The choice of l also gives that the $k_{\max}^{\text{th}}$ vector has $|(b_{k_{\max}}^{\text{out}}) - 1| > 2^{2r-1-\epsilon}$, or in other words a non-zero last entry. Also this particular last entry is larger than all of the other last entries.

It was a fact that multiplying all the last entries by a scalar can only increase G-S lengths so: $\| (b_i^{\text{in}})^* \| \leq \| (b_i^{\text{out}})^* \|$. So $\text{AD}^{\text{in}} \leq \text{AD}^{\text{out}}$.

It is also a fact that the product of G-S lengths is invariant under changes to basis ordering. So if we were to rearrange both bases (in and out) so that the $k_{\max}^{\text{th}}$ vector is now first then we would know that $\| (b_{k_{\max}}^{\text{in}})^* \| \leq r$ and $\| (b_{k_{\max}}^{\text{out}})^* \| \geq 2r - 1 - \epsilon$. So the product of the rearranged output vectors is still $= \text{AD}^{\text{out}} \geq 2^{r-1-\epsilon}\text{AD}^{\text{in}}$. Since the product of the rearranged input vectors is $= \text{AD}^{\text{in}}$ and $2^{2r-1-\epsilon} = 2^{r-1-\epsilon} \cdot 2^r$.

□

It remains to show Properties 6, 9 and 11

- **6.** $n_{\text{scales}} \leq 3r + 2$ needs to be checked in this section, so let's prove that $n_{\text{scales}}^{\text{in}} \leq 3r + 1$ which will verify that $n_{\text{scales}}^{\text{out}} \leq 3r + 2$ since n_{scales} only increased by one during this procedure.

Lemma 15. $n_{scales}^{in} \leq 3r + 1$

Proof. We know the $\mathrm{AD}^{\mathrm{in}} \leq 2^{r^2}$. We also know that every time n_{scales} increases by one AD increases by a factor $\geq 2^{r-1-\epsilon}$. An appeal to Theorems 4 in section **3.7**, and 5 in **3.9**, and $n_{\mathrm{rm}} \leq r + n_{\mathrm{good}} - 1$ reveals that $\mathrm{AD}^{\mathrm{in}} \geq 2^{(r-1)n_{\mathrm{good}} - 2r^2 + 2r}$ regardless of n_{scales}. So in the worst case $\mathrm{AD}^{\mathrm{in}} \geq 2^{(r-1-\epsilon)n^{\mathrm{in}}_{\mathrm{scales}} - 2r^2 + 2r}$ and thus $2^{3r^2 - 2r} \geq 2^{(r-1-\epsilon)n^{\mathrm{in}}_{\mathrm{scales}}}$ which implies $n^{\mathrm{in}}_{\mathrm{scales}} \leq 3r + 1.3$ for $\epsilon = .1$. Since r and $n^{\mathrm{in}}_{\mathrm{scales}}$ are both integers we have completed the proof. □

- **9.** $P^{\mathrm{out}} \geq P^{\mathrm{in}}$ follows from **12.** $\max\{\| (b_i^{\mathrm{out}}))^* \|\} \leq 2^{2r}$
- **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\mathrm{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r + 1$ for any irreducible factor g

Proof. For each v_g the lift of v_g into L^{out} has a last entry scaled by TS $\leq$ MSU so this entry is still smaller than $\frac{1}{\sqrt{3r^2+5r+5}}$. This is enough by observing Fact 2 in section 3.3.3. □

3.10.6 Properties true of Exit 16

In this case no vectors or variables have changed except c (if we keep that format) so

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\mathrm{out}}))^* \|\} \leq 2^r$
- **13.** $\mathrm{AD}^{\mathrm{out}} \leq 2^{r^2}$
- **14.** $\mathrm{AD}^{\mathrm{out}} = \mathrm{AD}^{\mathrm{in}}$

all remain true.

3.10.7 Complexity Notes:

- This Procedure is only called from **3.6(Check If Solved)** (Exit 14). This happens $\leq n_{\mathrm{novec}} + n_{\mathrm{good}} + n_{\mathrm{bad}} + n_{\mathrm{scales}}$ times. (See complexity notes in **3.6**.)
- If there was a successful scaling then we call **3.11(LLL)** (Exit 15), and n_{scales} increases every time this happens. So Exit 15 is called n_{scales} times.
- The other exit is a call to **3.8(Find Next Coeff)** (Exit 16) which therefore happens $\leq n_{\mathrm{novec}} + n_{\mathrm{good}} + n_{\mathrm{bad}}$ times.

3.11 LLL

3.11.1 Variables used in this Procedure

- n_{switches} counts the number of LLL switches which have taken place throughout the algorithm and only changes here and in **3.9(pLLL)**.
- Running LLL makes actual changes to the b_i but does not change L.

3.11.2 Procedure

1. Run LLL and let n_{switches} do its thing
2. Exit this procedure and call Remove Vectors (Exit 12)

3.11.3 What This Procedure Does:

This is any lattice reduction algorithm so perhaps we will be using B-reduce from before or maybe Stehlé's floating point LLL. It should be noted that AD is invariant under LLL switches. Also the maximum l_i cannot be increased (nor the minimum decreased), and if the second largest G-S length does increase it is at the expense of the largest. The output will of course be LLL reduced or B-reduced (defined in Section 2.1). n_{switches} is increased every time an LLL switch takes place.

3.11.4 Properties true of the Input

- To confirm these properties check **3.10(Scale Up)** (Exit 15) and **3.12(No Vector Added)** (Exit 10).
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^{2r}$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{3/2r^2+5/2r-1}$

3.11.5 Properties true at Exit 12

- $b_1, \ldots, b_s$ form an LLL reduced basis, not yet B-reduced, and $L^{\text{out}} = L^{\text{in}}$.
- $s, n_{\text{good}}, n_{\text{bad}}, n_{\text{novec}}, n_{\text{rm}}, n_{\text{scales}}, n_{\text{entries}}$ are unaffected so **1.** $s < \lfloor \frac{3r}{2} \rfloor$, **2.** $n_{\text{good}} \leq 3r + 2$, **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$, **4.** $n_{\text{novec}} \leq 3r + 2$, **5.** $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r + 1$, **6.** $n_{\text{scales}} \leq 3r + 2$, and **7.** $n_{\text{entries}} \leq 3r^2 + 5r + 5$ remain true.

- **8.** $n_{\text{switches}} \leq P$ follows from the fact that every switch increases Progress by at least 1 (proved already in Section **3.9**).

- **9.** $P^{\text{out}} \geq P^{\text{in}}$ follows from our proof of property **8**.

- **10.** $W \subseteq \pi(L)$ and **11.** $\| \pi^{-1}(v_g) \|^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r+1$ for any irreducible factor g, remain true because L was fixed

- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$ is true by LLL observation 1 in section 1.3.

- **13.** $\text{AD}^{\text{out}} \leq 2^{3r^2}$ and **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$ both true since AD invariant under LLL switches (although the bound is not important).

3.11.6 Complexity Notes:

- This procedure is called n_{novec} times from **3.12(No Vector Added)** (Exit 10) and n_{scales} times from **3.10(Scale Up)** (Exit 15).

- There is only one exit which is calling **3.7(Remove Vectors)** (Exit 12). Therefore this happens $n_{\text{scales}} + n_{\text{novec}}$ times.

- The complexity of this step is combined with the complexity of the LLL part of **3.9(pLLL)**. We bound n_{switches} and multiply this by the cost of a single switch. We then include the overhead cost of an LLL call times $n_{\text{novec}} + n_{\text{scales}}$ for this procedure and $n_{\text{good}} + n_{\text{bad}}$ for **3.9**.

3.12 Decide If Vector Added

3.12.1 Variables used in this Procedure

- ISD, ESD, CSU, MSU are all variables related to the scaling of the current coefficient and are initialized here.

- ISD stands for the Initial Scale Down. ISD is chosen so that $2^{3r-1} < \frac{p^a}{2^{\text{ISD}}} \leq 2^{3r}$.

- ESD is the Extra Scale Down we can do only in this step and it might not be needed, but is the primary difference between this scaling and the scaling in **3.10**. (It will scale down in the case that there is a new entry which is larger than $2^{2r-\epsilon}$, which doesn't happen in **3.10**.) It should be noted that we might need ESD to actually be a scale up (not down) here, but this doesn't change anything.

- CB$(c) + d$ was a quantity defined in **3.8(Find Next Coeff)**. This is the bound we use for making sure MSU the maximum scale up prevents us from scaling the entries by too much. This is what is needed to keep Property 11 true throughout the algorithm.

- MSU is the Maximum (available) Scale Up, it is used in **3.10**, but ensures that the scaling back up doesn't go too far (in which case we could violate property 11).

- CSU, the Current Scale Up, is initialized as 0 here for emphasis but doesn't come into play until **3.10**.

- $\mathrm{TS}_{\mathrm{tc}}$ is Total Scaling and is also initialized here. This number will be updated in **3.10**, and is a historical marker of how much scaling has been performed on what is currently the final $((\mathrm{tc} + r)^{\mathrm{th}})$ entry of the b_i. Since tc will change as we use new coefficients but $\mathrm{TS}_{\mathrm{tc}}$ will not then we can later refer to the total scaling done on the ith coefficient used.

- $k_{\max}$ was found in **3.8**. K is a local variable just used, for emphasis, as the size of the largest potential new entry.

- T and S (used here and in a proof during the bad coeff case in **3.9(pLLL)**) are designed to test when we need the $(\vec{0}, p^a)$ vector to ensure **10.** $W \subseteq \pi(L)$ (Exit 9), and when we don't (the No Vector Added Case, Exit 10).

- n_{novec} is increased only in this procedure and only when the No Vector Added Case happens (Exit 10).

- n_{entries} counts the number of entries in each b_i, and is increased in this procedure if it uses Exit 10or is increased in **3.9** otherwise.

- i is a local variable for a simple index.

3.12.2 Procedure

1. ISD := $\lceil \log_2(\frac{p^a}{2^{3r}}) \rceil$ (Initial Scale Down) (Note: $2^{3r-1} < \frac{p^a}{2^{\mathrm{ISD}}} \leq 2^{3r}$)

2. CSU := 0; (Current Scale Up)

3. ESD := 0; (Extra Scale Down)

4. MSU := ISD $-$ CB$(c) - d$; (Maximum Scale Up)

5. $\mathrm{TS}_{\mathrm{tc}}$:= ISD $-$ CSU; (Total Scaling so far on the $\mathrm{tc}^{\mathrm{th}}$ coefficient/new entry we have used.)

6. $K := |\psi_c^a(b_{k_{\max}})|$; ($k_{\max}$ was computed in **3.8(Find Next Coeff)**)

7. $T := \frac{p^a - 2^{\mathrm{ISD}} \cdot B}{K}$;

8. $S := B \cdot (2(\frac{3}{2})^{s-1} - 2)$;

9. **Probationary Case:** If $S \geq T$ then exit this procedure and call **3.9(pLLL)** (Exit 9)

10. **No Vector Added Case:** If $S < T$ then

    (a) $n_{\mathrm{novec}} := n_{\mathrm{novec}} + 1$;

    (b) $\mathrm{ESD} := \lceil \log_2(\frac{|\psi_c^a(b_{k_{\max}})|}{2^{2r-\epsilon}}) \rceil - \mathrm{ISD}$; (Extra Scale Down)

    (Note: $2^{r-1-\epsilon} \parallel b_{k_{\max}} \parallel \leq 2^{2r-1-\epsilon} < \frac{|\psi_c^a(b_{k_{\max}})|}{2^{\mathrm{ESD}+\mathrm{ISD}}} \leq 2^{2r-\epsilon}$.)

    (c) $\mathrm{MSU} := \mathrm{MSU} + \mathrm{ESD}$ (New Max Scale Up)

    (d) $\mathrm{TS}_{\mathrm{tc}} := \mathrm{ISD} + \mathrm{ESD} - \mathrm{CSU}$ (New Total Scaling of $\mathrm{tc}^{\mathrm{th}}$ new entry)

    (e) For i to s do

    $b_i := (b_i, \frac{\psi_c^a(b_i)}{2^{\mathrm{ESD}+\mathrm{ISD}}})$

    od;

    (f) $n_{\mathrm{entries}} := n_{\mathrm{entries}} + 1$

    (g) Exit this procedure and call **3.11(LLL)** (Exit 10)

    fi;

3.12.3 What This Procedure Does:

In general, when we add a new entry, we also need to add a vector of the form $(0, \ldots, 0, *)$ to ensure that property 11 remains true (so that **10.** $W \subseteq \pi(L)$ will hold during **3.7(Remove Vectors)**). In the very rare case that $S < T$ is true, we can ensure that 11 remains true even if we do not add the vector $(0, \ldots, 0, *)$. By not inserting the $(0, \ldots, 0, *)$ vector at the beginning of $b_1, \ldots, b_s$ we can save a small number (namely $\mathcal{O}(r)$) of LLL switches. Saving these switches means a small improvement to the algorithm, but not one that is likely to have any observable impact in practice. However, knowing that $S \geq T$ when we call **3.9** greatly simplifies our complexity proof.

If we don't need the $(0, \ldots, 0, *)$ vector then we then can perform a scaling on the new entry before calling **3.11(LLL)**. It should be noted that all of the scaling which takes place in this procedure is encapsulated in the choice of ESD which ensures that no vector can be smaller than $2^{2r-1-\epsilon}$ in G-S length, and no vector can have G-S length larger than 2^{2r}.

In the likely event that $S \geq T$ we use Exit 9, and this procedure has done almost nothing other than define $k_{\max}$ and decided that $S \geq T$.

3.12.4 Properties true of the Input

- To confirm these properties check **3.8(Find Next Coeff)** (Exit 8).
- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)
- Properties **1** through **11** are all true
- **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$
- **13in.** $\text{AD}^{\text{in}} \leq 2^{r^2}$
- $\text{Avail_Bits}(c) \geq 3r$
- There is a large potential entry:
  $|\psi_c^a(b_{k_{\max}}^{\text{in}})| > 2^{2r} \cdot 2^{\text{CB}(c)+d} \geq 2^r \cdot \| b_{k_{\max}}^{\text{in}} \| \cdot 2^{\text{CB}(c)+d}$

3.12.5 Properties true at Exit 10

- $s, n_{\text{good}}, n_{\text{bad}}, n_{\text{rm}}, n_{\text{scales}}$ have not changed so **1.** $s < \lfloor \frac{3r}{2} \rfloor$, **2.** $n_{\text{good}} \leq 3r + 2$, **3.** $n_{\text{bad}} \leq 3r^2 - 2r + 1$, **5.** $n_{\text{rm}} \leq r + n_{\text{good}} - 1 \leq 4r + 1$, and **6.** $n_{\text{scales}} \leq 3r + 2$ still remain true.
- **8.** $n_{\text{switches}} \leq P$ just requires showing that **9.** $P^{\text{out}} \geq P^{\text{in}}$ since n_{switches} is unaffected. But all we have done is add an entry to each vector which can only increase G-S lengths, not decrease them. So P can only go up and certainly not down.
- **10.** $W \subseteq \pi(L)$ remains true since $\pi(L)$ is unaffected by adding new entries to the b_i.
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^{2r}$:

  Proof. We are given **12in.** $\max\{\| (b_i^{\text{in}}))^* \|\} \leq 2^r$ and that $b_1^{\text{in}}, \ldots, b_s^{\text{in}}$ is a B-reduced set. So in particular we know that $\| b_i \| \leq 2^r$ (actual lengths).

  There is a new last entry for each b_i, $|(b_i^{\text{out}})_{-1}|$, and we chose ESD so that $|(b_i^{\text{out}})_{-1}| \leq 2^{2r-\epsilon}$. So $\| b_i^{\text{out}} \| = \sqrt{(\| b_i^{\text{in}} \|)^2 + |(b_i^{\text{out}})_{-1}|^2}$

  Which is $\leq \sqrt{2^{2r} + 2^{4r-2\epsilon}} = 2^{2r}(\sqrt{\frac{1}{2^{2r}} + \frac{1}{2^{2\epsilon}}}) \leq 2^{2r}$.

  These are actual lengths which implies that $\| b_i^* \| \leq 2^{2r}$. □

- **13.** $\mathrm{AD}^{\mathrm{out}} \leq 2^{3r^2}$ is a rough bound from using **12.** $\max\{\| (b_i^{\mathrm{out}}))^* \|\} \leq 2^{2r}$ and **1.** $s < \lfloor \frac{3r}{2} \rfloor$. (It plays no important role.)

- **14.** $\mathrm{AD}^{\mathrm{out}} \geq 2^{r-1-\epsilon} \cdot \mathrm{AD}^{\mathrm{in}}$

  Proof. We want to investigate $\mathrm{AD}^{\mathrm{out}}/\mathrm{AD}^{\mathrm{in}}$. Here we want to take advantage of the big vector that never got added, since a vector of the form $(\vec{0}, \ldots)$ keeps the new G-S lengths from changing.

  So let $w_1 := (\vec{0}, \frac{p^a}{2^{\mathrm{ESD+ISD}}}), w_2 := b_1^{\mathrm{out}}, \ldots, w_{s+1} := b_s^{\mathrm{out}}$, and w_i^* the corresponding G-S vectors. Note that w_i^* and $(b_{i-1}^{\mathrm{in}})^*$ have all the same entries except w_i^* has an extra 0 as a last entry ($i \neq 1$). So $\prod_{i=1}^{s+1} \| w_i^* \| = \mathrm{AD}^{\mathrm{in}} \cdot \frac{p^a}{2^{\mathrm{ISD+ESD}}}$. But products of G-S lengths are invariant under basis ordering so this would still be true if we rearranged w_i as $v_2 := w_2, \ldots, v_{s+1} := w_{s+1}, v_1 := w_1$ now let the G-S orthogonalization of this rearranged set be defined as v_i^*.

  But in that ordering $\prod_2^{s+1} \| v_i^* \| = \prod \| (b_i^{\mathrm{out}})^* \| = \mathrm{AD}^{\mathrm{out}}$.

  So if we knew the G-S length of the big vector when it's the last vector then we know how much the AD has increased from beginning to end. Or in other words if we let v_1^* denote the G-S vector of v_1 when it is the last vector in our basis we see: $\mathrm{AD}^{\mathrm{in}} \cdot (\frac{p^a}{2^{\mathrm{ISD+ESD}}}) = \mathrm{AD}^{\mathrm{out}} \cdot \| v_1^* \|$. So $\mathrm{AD}^{\mathrm{out}}/\mathrm{AD}^{\mathrm{in}} = \frac{p^a}{2^{\mathrm{ISD+ESD}} \cdot \| v_1^* \|}$.

  When investigating $\| v_1^* \|$ it should be noted that $\| v_1^* \| \leq \| v_1 - x \cdot b_j^{\mathrm{out}} \|$ for any j and any $x \in \mathbb{R}$. In particular if we choose $j = k_{\max}$ and $x = \frac{p^a}{2^{\mathrm{ISD+ESD}} \cdot (b_{k_{\max}}^{\mathrm{out}})_{-1}}$ then the last terms will cancel and $\| v_1 - x \cdot b_j^{\mathrm{out}} \| = \frac{p^a \| b_{k_{\max}}^{\mathrm{in}} \|}{2^{\mathrm{ISD+ESD}} \cdot |(b_{k_{\max}}^{\mathrm{out}})_{-1}|}$.

  But the choice of ESD made sure that this is $\leq \frac{p^a}{2^{\mathrm{ISD+ESD}} \cdot 2^{r-1-\epsilon}}$

  So overall we have $\| v_1^* \| \leq \frac{p^a}{2^{\mathrm{ISD+ESD}} \cdot 2^{r-1-\epsilon}}$ which implies that $\frac{\mathrm{AD}^{\mathrm{out}}}{\mathrm{AD}^{\mathrm{in}}} \geq 2^{r-1-\epsilon}$ proving **14.** $\mathrm{AD}^{\mathrm{out}} \geq 2^{r-1-\epsilon} \cdot \mathrm{AD}^{\mathrm{in}}$. □

It remains to prove properties **4, 7,** and **11**.

- **4.** $n_{\mathrm{novec}} \leq 3r + 2$ needs to be checked in this section, so let's prove that $n_{\mathrm{novec}}^{\mathrm{in}} \leq 3r + 1$ which will verify that $n_{\mathrm{novec}}^{\mathrm{out}} \leq 3r + 2$ since n_{novec} only increased by one during this procedure.

  Lemma 16. $n_{novec}^{in} \leq 3r + 1$

  Proof. We know the $\mathrm{AD}^{\mathrm{in}} \leq 2^{r^2}$. We also know that every time n_{novec} increases by one AD increases by a factor $\geq 2^{r-1-\epsilon}$. An appeal to Theorems 4 in section **3.7**, and 5 in **3.9**, and $n_{\mathrm{rm}} \leq r + n_{\mathrm{good}} - 1$ reveals that $\mathrm{AD}^{\mathrm{in}} \geq 2^{(r-1)n_{\mathrm{good}} - 2r^2 + 2r}$ regardless of n_{novec}. So in the worst case $\mathrm{AD}^{\mathrm{in}} \geq 2^{(r-1-\epsilon)n_{\mathrm{novec}}^{\mathrm{in}} - 2r^2 + 2r}$

and thus $2^{3r^2-2r} \geq 2^{(r-1-\epsilon)n^{\text{in}}_{\text{novec}}}$ which implies $n^{\text{in}}_{\text{novec}} \leq 3r+1.3$ for $\epsilon = .1$. Since r and $n^{\text{in}}_{\text{novec}}$ are both integers we have completed the proof. □

- **7.** $n_{\text{entries}} \leq 3r^2+5r+5$ now follows from **4.** $n_{\text{novec}} \leq 3r+2$, **2.** $n_{\text{good}} \leq 3r+2$, **3.** $n_{\text{bad}} \leq 3r^2-2r+1$, and fact that $n_{\text{entries}} = r + n_{\text{novec}} + n_{\text{good}} + n_{\text{bad}}$.

- **11.** $\parallel \pi^{-1}(v_g) \parallel^2 \leq r + (n_{\text{entries}}) \cdot (\frac{1}{\sqrt{3r^2+5r+5}})^2 \leq r+1$ for any irreducible factor g requires a rather long proof:

Proof. With Fact 2 and Observation 2 from 3.3.3 and the fact that n_{entries} increased by one, we just need to show that $|(\pi^{-1}(v_g))_{-1}| = \frac{\text{Coeff}^a_c(g)}{2^{\text{TS}}}$ since $\text{TS} > \text{CB}(c)+d$. Then this new entry will be small enough to satisfy the property.

We know that $v_g \in \pi(L)$ for every g and that our new entry ensures that $(\pi^{-1}(v_g))_{-1} = \frac{\psi^a_c(v_g)}{2^{\text{ISD}}}$, but if we also include the vector $v = (\vec{0}, \frac{p^a}{2^{\text{ISD}}})$ in L, then $(\pi^{-1}(v_g))_{-1} = \frac{\text{Coeff}^a_c(v_g)}{2^{\text{ISD}}}$ and $\parallel \pi^{-1}(v_g) \parallel \leq B$.

So if $L = \text{SPAN}_{\mathbb{Z}}\{b^{\text{out}}_1, \ldots, b^{\text{out}}_s, v\}$ then we have $\parallel \pi^{-1}(v_g) \parallel \leq B$ with $\pi^{-1}(v_g) \in L$. Now I claim that $\parallel v^* \parallel > B$, in which case $\pi^{-1}(v_g) \in L^{\text{out}} = \text{SPAN}_{\mathbb{Z}}\{b^{\text{out}}_1, \ldots, b^{\text{out}}_s\}$ satisfying the property.

So all that remains is to prove the claim:

CLAIM: $\parallel v^* \parallel > B$

Proof of Claim:

By observation of G-S process we know that there are $c_i \in \mathbb{R}$ with $v^* = (\vec{0}, \frac{p^a}{2^{\text{ISD}}}) - \sum_{i=1}^s c_i(b^{\text{in}}_i, \frac{\psi^a_c(b^{\text{in}}_i)}{2^{\text{ISD}}})$ there are two cases:

Case 1: $\sum_{i=1}^s |c_i| < \frac{p^a - 2^{\text{ISD}}B}{|\psi^a_c(b^{\text{in}}_{k_{\max}})|}$

In this case we will show that the last component of v^* has size $> B$.

$$B \cdot 2^{\text{ISD}} < p^a - (\sum |c_i|) \cdot |\psi^a_c(b^{\text{in}}_{k_{\max}})| \leq p^a - (\sum c_i \psi^a_c(b^{\text{in}}_i))$$

which implies:

$$(v^*)_{-1} = \frac{p^a}{2^{\text{ISD}}} - (\sum c_i \frac{\psi^a_c(b^{\text{in}}_i)}{2^{\text{ISD}}}) > B$$

which gives: $\parallel v^* \parallel > B$

Case 2: $\sum_{i=1}^s |c_i| \geq \frac{p^a - 2^{\text{ISD}}B}{|\psi^a_c(b_{k_{\max}})|}$

In this case we show that the length of $\parallel \sum c_i b^{\text{in}}_i \parallel > B$ which is size of all but the last entry of v^*.

Observe: $b_i^{\text{in}} = (b_i^{\text{in}})^* + \sum_{j<i} \mu_{i,j}(b_j^{\text{in}})^*$ for all i, and since LLL reduced we know $|\mu_{i,j}| \le \frac{1}{2}$.

$$\sum_{i=1}^{s} c_i b_i = \sum_{i=1}^{s} c_i(b_i^* + \sum_{j<i} \mu_{i,j} b_j^*) = \sum_{i=1}^{s}(c_i + \sum_{k>i} \mu_{k,i} c_k) b_i^*$$

and b_i^* are pairwise orthogonal with lengths ≥ 1 so $\| \sum_{i=1}^{s}(c_i b_i) \| \ge \sqrt{\sum_{i=1}^{s} (c_i + \sum_{k>i} \mu_{k,i} c_k)^2}$.

So if there is at least one i with $|c_i + \sum_{k>i} \mu_{k,i} c_k| > B$ then we are done.

Assume that there is no such i.

Then $|c_s| \le B$ and $|c_{s-1} + \mu_{s,s-1} c_s| \le B$ so $|c_{s-1}| \le B \cdot \frac{3}{2}$ (since $|\mu| \le 1/2$).

This leads to an inductive argument which shows: $|c_{s-i}| \le B \cdot (\frac{3}{2})^i$.

We know $|c_{s-i}| \le B \cdot (3/2)^i$ works for $i = 0, 1$.

Assume this is true for all $i < k$.

We know that $|c_{s-k} + \mu_{s-(k-1),s-k} c_{s-(k-1)} + \cdots + \mu_{s,s-k} c_s| \le B$

Supposing that all $\mu_{s-i,s-k} = (-1/2)$ and $|c_{s-i}| \le B \cdot (\frac{3}{2})^i$ shows $|c_{s-k}| \le (\frac{1}{2}) \cdot B \cdot (\frac{3}{2})^{k-1} + \cdots + (\frac{1}{2}) \cdot B \cdot (\frac{3}{2})^1 + (\frac{1}{2}) \cdot B + B = B(\frac{3}{2})^k$.

Therefore $\sum_{i=1}^{s} |c_i| = \sum_{i=0}^{s-1} |c_{s-i}| \le B \cdot (\sum_{i=0}^{s-1} (\frac{3}{2})^i)$ but $S < T$ and the case 2 criteria show that $B \cdot (\sum_{i=0}^{s-1} (\frac{3}{2})^i) < \frac{p^a - 2^{\text{ISD}} B}{|\psi_c^a(b_{k_{\max}})|} \le \sum_{i=1}^{s} |c_i|$ Which gives a contradiction. So there must have been at least one i with $|c_i + \sum_{k>i} \mu_{k,i} c_k| > B$

$\Rightarrow \| v^* \| > B$ This concludes the proof of the claim.

End Proof of Claim.

Now we know that the property holds just before the extra scale down, so we need to show that ESD doesn't scale down by too much so that $\text{ISD} + \text{ESD} = \text{TS}_{\text{tc}} \ge \text{CB}(c) + d$ after the procedure.

The choice of ESD shows us that $\frac{\psi_c^a(b_{k_{\max}})}{2^{2r-\epsilon}} \le 2^{\text{ESD}+\text{ISD}}$.

But we know from the properties of the input that $2^{2r} 2^{\text{CB}(c)+d} \le \psi_c^a(b_{k_{\max}})$.

Putting these together we see that $2^{\text{CB}(c)+d+\epsilon} \le 2^{\text{ISD}+\text{ESD}}$. □

3.12.6 Properties true at Exit 9

No important variables changed and no vectors changed so the input properties remain true.

- $b_1, \ldots, b_s$ is B-reduced (defined in Section 2.1)

- Properties **1** through **11** are all true
- **12.** $\max\{\| (b_i^{\text{out}}))^* \|\} \leq 2^r$
- **13.** $\text{AD}^{\text{out}} \leq 2^{r^2}$
- **14.** $\text{AD}^{\text{out}} = \text{AD}^{\text{in}}$
- $\text{Avail_Bits}(c) \geq 3r$
- There is a large potential entry:
  $|\psi_c^a(b_{k_{\max}}^{\text{in}})| > 2^{2r} \cdot 2^{\text{CB}(c)+d} \geq 2^r \cdot \| b_{k_{\max}}^{\text{in}} \| \cdot 2^{\text{CB}(c)+d}$
- We also learned that $S \geq T$

3.12.7 Complexity Notes:

- This procedure has two exits one of them is **3.11(LLL)** (Exit 10) which only happens in the No Vector Added Case so n_{novec} times.
- The other exit is **3.9(pLLL)** (Exit 9) which happens $n_{\text{good}} + n_{\text{bad}}$ times (see complexity notes **3.9**).
- Since this procedure is only called from one place (**3.8(Find Next Coeff)** Exit 8) this can only happen $n_{\text{novec}} + n_{\text{good}} + n_{\text{bad}}$ times.
- The complexity of this procedure is small compared to the overall complexity of the algorithm.

3.13 Zassenhaus' Algorithm

Run the Standard Zassenhaus Algorithm on f with p.

Call Output with a complete irreducible factorization of f over the rationals. (Exit 2)

Note that this will be efficient since $r < 10$.

3.14 Switch Complexity of this Algorithm

Since **8.** $n_{\text{switches}} \leq P$ and **9.** $P^{\text{out}} \geq P^{\text{in}}$ hold throughout the entire algorithm and $P \leq 68r^3$ when the algorithm outputs during **3.6**, then the number of switches made throughout the entire algorithm is $\leq 68r^3$. Thus the switch-complexity is now $\mathcal{O}(r^3)$, independent of both degree and coefficient size. The switch complexity is the dominant term of the van Hoeij and LLL factoring algorithms' complexities. We don't have time to compute the actual complexity before this publication but we have provided a count of how often each step is called as a step toward the overall complexity.

CHAPTER 4

Why this thesis is interesting

The switch complexity of $\mathcal{O}(Nr^2)$ in Chapter 2 is already a pretty interesting result, and Chapter 3 is long and technical for such a small improvement in switch complexity to $\mathcal{O}(r^3)$. So why is it worth it? Because now for the first time since the early 1980's the fastest algorithm in practice and the fastest algorithm in theory are one and the same, which would not be true had we stopped. Seeing why this algorithm is actually faster than van Hoeij's algorithm (in many cases, and certainly never slower) is a matter of exploring the new practical feature: Early Termination.

4.1 Early Termination

We recall the Key Practical Problem: Suppose we want to factor a polynomial f and that Hensel Lifting to accuracy p^{100} is enough to solve the combinatorial problem and reconstruct all of the factors. Well with the quadratic Hensel Lifting we perform we would reach p^{128} before solving the problem which is near optimal. Recall the key practical problem from 1.2.2 for effective early termination algorithms:

Shouldn't it waste CPU time to stop after p^{32} and p^{64} and search through coefficients and try solving the combinatorial problem? Why wouldn't it be faster to lift to p^{128} immediately and begin trying there?

Because of the design of our algorithm we ensured that we only ever take steps which lead to an increase in Progress, and when Progress reaches its bound the algorithm will be stopped. So it didn't matter where the data came from and the work done during early Hensel Steps is never wasted.

So now the two basic problems with Zassenhaus' algorithm from Chapter 1 have been resolved:

- The exponential search through local factors is now fast because of van Hoeij's approach.
- The possibility of Hensel Lifting too far (if the bound on coefficients of factors is too pessimistic) is resolved by this Early Termination algorithm.

What's more is that both practical problems are resolved by an algorithm with good theoretical complexity! So we have resolved our key theoretical problem from section 1.4.2.

4.1.1 Good cases for Early Termination

We've now seen that the Early Termination algorithm will never slow down the algorithm, but when will it speed up the algorithm? In the case that f is a large irreducible multiplied by any number of smaller polynomials (including f being irreducible) we will be able to reconstruct the smaller factors with much less Hensel Lifting than it takes to reconstruct the large factor. In this case we divide away the small factors and if we know that what remains is a single factor than we don't need to reconstruct it since we know f and all but one factor so division will provide the final factor. In the case that f is irreducible this just amounts to the fact that solving the combinatorial problem often requires less Hensel Lifting than is used when Hensel Lifting to the bound in Zassenhaus.

4.2 Conclusion

Ever since LLL (1983) there has been a gap between the best factoring algorithm in theory, and the best algorithm in practice. Before 2001, the Zassenhaus algorithm performed best in practice, while LLL/Schönhage had the best theoretical complexity. This gap between theory and practice grew even wider with [6] because this algorithm was even faster in practice, while even worse in theory ([6] contains no complexity bound).

Then in [3] the gap was made smaller; polynomial time complexity bounds for two versions of the [6] algorithm were given. However, the gap between theory and practice remained very large because the version that was faster in practice received the worse theoretical bound!

We have now resolved this unfortunate situation. In this thesis we have made the (in practice) fastest version even faster, in practice, by saving time on Hensel lifting. We have proved a bound for the switch-complexity that is *asymptotically sharp*. This bound perfectly captures the actual behavior of the algorithm, in fact, the progress P could even be used to give a realistic progress bar!

We do not merely reduce the gap between theory and practice; we eliminate the gap altogether. *The algorithm that is best in practice, and the algorithm that is best in theory, will now be the same algorithm.*

Our theoretical work resulted in more than just a better bound for the complexity of factoring.

It also allowed us to solve the key practical problem (mentioned earlier) in designing an efficient early termination algorithm. Recall that the key problem was wasting time on attempts that were "unsuccessful" because we had not lifted far enough.

4.3 Future Improvements

To make to this algorithm even faster in practice there is another improvement we can make. The scalings performed throughout this algorithm leave us with rational entries in our vectors, where as, if these entries were integers we could perform the LLL switches faster. The problem is fairly easily resolved by rounding the entries to something in $2^{-k}\mathbb{Z}$ for a fixed k and adjusting the entries occasionally. But the question is whether the error terms can decrease P. Well it does decrease P a little bit, but only by a fixed amount so long as we re-round every so often. This improvement is relatively easy to deal with and has no effect on the theoretical complexity other than maybe a slightly larger constant term in switch complexity (but a smaller term in cost per switch).

Also we haven't dealt with the total complexity of the algorithm yet, only the switch complexity since it was the dominant term in prior algorithms.

REFERENCES

[1] J. Abbott, V. Shoup and P. Zimmermann, *Factorization in* $\mathbb{Z}[x]$*: The Searching Phase*, ISSAC'2000 Proceedings, 1–7 (2000).

[2] K. Belabas *A relative van Hoeij algorithm over number fields*, J. Symbolic Computation, **37** (2004), pp. 641–668.

[3] K. Belabas, M. van Hoeij, J. Klüners, and A. Steel, *Factoring polynomials over global fields*, preprint arXiv:math/0409510v1 (2004).

[4] E. R. Berlekamp, *Factoring polynomials over finite fields*, Bell System Technical Journal **46**, 1853-1859, (1967).

[5] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra 1st ed.*, Cambridge University Press, (1999).

[6] M. van Hoeij, *Factoring polynomials and the knapsack problem*, J. Number Theory, **95** (2002), pp. 167–189.

[7] E. Kaltofen, *Polynomial factorization.* In: Computer Algebra, 2nd ed,, editors B. Buchberger et all, Springer Verlag, 95–113 (1982).

[8] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), pp. 515–534.

[9] P. Nguyen, and D. Stehlé, *Floating-Point LLL Revisited*, Lecture Notes in Computer Science, Proceedings of Eurocrypt 2005, Springer-Verlag, Vol. 3494 pp. 215-233.

[10] A. Schönhage, *Factorization of univariate integer polynomials by Diophantine approximation and an improved basis reduction algorithm*, Proc. ICALP 84, Springer Lec. Notes Comp. Sci. **172**, (1984), pp. 436–447.

[11] H. Zassenhaus, *On Hensel factorization I*, Journal of Number Theory (1969), pp. 291–311.

BIOGRAPHICAL SKETCH

Andy Novocin

Andy was born June 12, 1983 in Jacksonville, Florida. He met Mark van Hoeij in 2001, shortly after high school, and has worked with Mark ever since. There is very little in the world that doesn't interest Andy, making mathematics a very natural choice. He will never turn down a game of Scrabble or Puerto Rico.