

Complexity analysis of Polynomial Factorization

Mark van Hoeij^{1*}, Andrew Novocin², and William Hart³

¹ Florida State University, 208 Love Building Tallahassee, FL 32306-4510

² Laboratoire LIP (U. Lyon, CNRS, ENS Lyon, INRIA, UCBL), 46 Allée d'Italie, 69364 Lyon Cedex 07, France.

³ Warwick Mathematics Institute

Abstract. We present a comprehensive complexity analysis of an efficient algorithm for polynomial factorization in $\mathbb{Z}[x]$.

* Supported by NSF 0728853

This document is a work in progress. The central problem we address is the factorization of a square-free and monic polynomial in $\mathbb{Z}[x]$. Forgoing for the moment any introduction to the method or explanation of the reasoning we present the algorithm essentially as we have implemented it in FLINT.

We have a couple notations, M refers to the full M which may or may not be virtual depending on the boolean `virtual` (if it is virtual then pertinent data is in M and G). Also we'll use $\lfloor a/b \rfloor$ to refer to rounding but not into \mathbb{Z} but rather $\mathbb{Z}/2^r$.

Algorithm 1 *The main algorithm*

Input: Square-free, monic, polynomial $f \in \mathbb{Z}[x]$ of degree N

Output: The irreducible factors of f over \mathbb{Z}

1. Choose a prime, p , such that $\gcd(f, f') \equiv 1$ modulo p .
2. **Modular Factorization:** Factor f modulo $p \equiv f_1 \cdots f_r$.
3. **if** $r \leq Z_{\text{bound}} := 15$ **return** Zassenhaus(f)
4. Compute first target precision a with Algorithm 5
5. **until solved:**
 - (a) **Hensel Lifting:** Hensel lift $f_1 \cdots f_r$ to precision p^a .
 - (b) **Recombination:** Algorithm 2(f, f_1, \dots, f_r, p^a)
 - (c) **if** not solved: $a := 2a$

Algorithm 2 *Attempt Reconstruction*

Input: f, f_1, \dots, f_r the lifted factors, their precision p^a , and possibly $M \in \mathbb{Z}_{s \times (r+c)}$.

Output: If solved then the irreducible factors of f over \mathbb{Z} otherwise an updated M .

1. If this is the first call let $M := I_{r \times r}$
2. Choose a k heuristically (see below for details)
3. For $j \in \{0, \dots, k-1, N-k-1, \dots, N-1\}$ do:
 - (a) Compute CLD bound, X_j , for x^j using Algorithm 4
 - (b) If $\sqrt{(E_{\text{bound}} := 1.6r^2) \cdot X_j} \leq p^a / 2^{(\ell:=1.5)r}$ then compute new column vector $\mathbf{x}_j := (x_{1,j}, \dots, x_{r,j})^T$ where $x_{i,j}$ is the coefficient of x^j in $f \cdot f'_i / f_i$
4. For each computed \mathbf{x}_j do:
 - (a) `justified := True`; While `justified` is True do:
 - i. Decide if LLL is justified using Algorithm 3 which augments M
 - ii. If `justified` then: if `virtual` run G-LLL(M, G) else run LLL(M) both with parameters $\delta = .99, \eta = .51$
 - iii. Compute G-S lengths of rows of M
 - iv. Decrease the number of rows of M until the final Gram-Schmidt norm $\leq \sqrt{(B(r) := r+2)}$
 - v. Use Algorithm 6 to test if solved

Algorithm 3 *Decide if column is worth calling LLL*

Input: $M \in \mathbb{Z}_{s \times (r+c)}$, data vector \mathbf{x}_j, p^a, X_j the CLD bound for x^j , boolean `virtual`

Output: A potentially updated M and/or G and a boolean `justified`, and boolean `virtual`

1. Let $B := r+2$ and s be the number of rows of M
2. If $p^a < X_j \cdot B \cdot \sqrt{(E_{\text{bound}} = 1.6r^2) \cdot 2^{(\ell:=1.5)r}}$ `justified := False` and exit
3. Find U the first r columns of M
4. Compute $\mathbf{y}_j := U \cdot \mathbf{x}_j$

5. If $\| \mathbf{y}_j \|_\infty < X_j \cdot B \cdot \sqrt{(E_{\text{bound}} = 1.6r^2)} \cdot 2^{(\text{NoVec}_{\text{bound}} := .937r - .1)}$ then `justified := False` and exit
6. Find new column scaling $k := \lceil \log_2 \left(\frac{p^a}{2^{(\ell=1.5)r}} \right) \rceil$.
7. Embed $\mathbf{x}_j/2^k$ and $p^a/2^k$ into $\mathbb{Z}/2^r$ by rounding and denote results as $\tilde{\mathbf{x}}_j$ and \tilde{P}
8. Compute $\tilde{\mathbf{y}}_j := U \cdot \tilde{\mathbf{x}}_j$
9. If $\frac{\tilde{P} - \sqrt{B(r)}}{\|\tilde{\mathbf{y}}_j\|_\infty} > \sqrt{B(r)} (\sum_{i=0}^{s-1} (1 + \eta)^i)$ then:
 - (a) `no_vec := True`
 - (b) $k := \lceil \log_2 \left(\frac{\|\mathbf{y}_j\|_\infty}{2^{(\text{NoVec}_{\text{bound}} = .937r - .0001)}} \right) \rceil$
 - (c) Re-Embed $\mathbf{x}_j/2^k$ into $\mathbb{Z}/2^r$
 - (d) Compute $\tilde{\mathbf{y}}_j := U \cdot \tilde{\mathbf{x}}_j$
10. else `no_vec := False`
11. If `virtual == False` and Number of Columns of $M < 5r$ then:
 - (a) If `no_vec` is True then augment M with new column $\tilde{\mathbf{y}}_j = U \cdot \tilde{\mathbf{x}}_j$
If `no_vec` False then also adjoin a new row so

$$M := \left[\begin{array}{c|c} \mathbf{0} & \tilde{P} \\ \hline M & \tilde{\mathbf{y}}_j \end{array} \right]$$
 - (b) `Justified := True; return M`
12. if Number of Columns of M is $\geq 5r$ and `virtual == False` then:
 - (a) `virtual := True`
 - (b) Define $G \in \frac{\mathbb{Z}}{2^{2r}}_{s \times s}$ with $G[i, j] := \langle M[i], M[j] \rangle$ the exact Gram matrix of the now virtual M .
 - (c) Strip M to its first r columns.
13. if `virtual == True` then:
 - (a) If `no_vec` is True then update G with new ‘column’, $G[i, j] := G[i, j] + \tilde{\mathbf{y}}_j[i] \cdot \tilde{\mathbf{y}}_j[j]$ for all i, j
 - (b) If `no_vec` False then also adjoin a new ‘row’ so

$$M := \left[\begin{array}{c} \mathbf{0} \\ \hline M \end{array} \right]$$
 add ‘column’ $G[i, j] := G[i, j] + \tilde{\mathbf{y}}_j[i] \cdot \tilde{\mathbf{y}}_j[j]$ for $1 \leq i, j \leq s$ and then expand G to $(s+1) \times (s+1)$ via $G[i+1, j+1] := G[i, j]$ and $G[1, i+1] := G[i+1, 1] := \tilde{P} \cdot \tilde{\mathbf{y}}_j[i]$ for $1 \leq i, j \leq s$ and finally $G[1, 1] := \tilde{P}^2$.
 - (c) `Justified := True; return M, G`

Algorithm 4 CLD bound

Input: $f = a_0 + \dots + a_N x^N$ and $c \in \{0, \dots, N-1\}$

Output: X_c , a bound for the absolute value of the coefficient of x^c in the polynomial fg'/g for any $g \in \mathbb{Z}[x]$ dividing f .

1. Let $B_1(r) := \frac{1}{r^{c+1}} (|a_0| + \dots + |a_c| r^c)$
2. Let $B_2(r) := \frac{1}{r^{c+1}} (|a_{c+1}| r^{c+1} + \dots + |a_N| r^N)$
3. Find $r \in \mathbb{R}^+$ such that $\text{MAX}\{B_1(r), B_2(r)\}$ is minimized to within a constant.
4. return $X_c := N \cdot \text{MAX}\{B_1(r), B_2(r)\}$

For finding an r which minimizes $\text{MAX}\{B_1(r), B_2(r)\}$ our floating-point method is as follows (where $\text{sign}(x)$ is 1 if x positive, -1 if x negative, and 0 otherwise).

1. Let $r := 1$, `scaling := 2`, `cur_sign := sign(B1(r) - B2(r))`, and `pos_ratio := $\left(\frac{B_1(r)}{B_2(r)} \right)^{\text{cur_sign}}$`

2. Until `cur_sign` changes or `pos_ratio` ≤ 2 do:
 - (a) $r := r \cdot \text{scaling}^{\text{cur_sign}}$
 - (b) `cur_sign` := $\text{sign}(B1(r) - B2(r))$
 - (c) `pos_ratio` := $\left(\frac{B1(r)}{B2(r)}\right)^{\text{cur_sign}}$
3. If `pos_ratio` > 2 then `scaling` := $\sqrt{\text{scaling}}$ and Go to step 2 Otherwise Return r .

Algorithm 5 *Heuristic for initial precision*

Input: $f \in \mathbb{Z}[x]$, p

Output: Suggested target precision a

1. Use Algorithm 4 to compute b , the minimum of (CLD bound for x^0) and (CLD bound for x^{N-1}).
2. return $a := \left\lceil \frac{(7\ell/3=3.5)r + \log_2 b + (\log_2 E_{\text{bound}}=1.6r^2)/2}{\log_2 p} \right\rceil$

Algorithm 6 *Check if solved*

Input: M , f , f_1, \dots, f_r to precision p^a

Output: A Boolean, *solved*, and possibly the irreducible factors of f in $\mathbb{Z}[x]$

1. Sort the first r columns of M into classes of columns which are identical
2. If there are not more classes than there are rows of M then we have a potential solution otherwise `solved` := *False* and exit
3. For each class multiply the p -adic polynomials corresponding with the columns in that class and reduce with symmetric remainder modulo p^a to find the potential factors
4. In order, from the lowest degree to the highest degree, perform trial divisions of f
5. If any two polynomials fail to divide f then `solved` := *False* and exit
6. `solved` := *True* if there is one failed polynomial then recover it by division of f by the successful factors

Note that the virtualization of M has no impact on any of the following, the only entries of M which are needed exactly are either inner products or the first r columns, both of which are kept exactly. Decisions made by fpLLL are made from the exact Gram matrix not directly from M while updates of M by fpLLL should update both G and the stripped down $M == U$. This should never be needed in practice but in the worst-case analysis it is possible that r^2 entries could be included in M and we can provide a better complexity using this ‘virtual entries’ method. It should be noted that this analysis is simple in fpLLL but in our situation (when the inputs have about the same number of bits as the dimension) this virtual method can result in a slightly lower ‘overhead’ for fpLLL.

We adopt the standard theoretical goals: A) To give a complete complexity bound of the above algorithm for f of degree N , r local factors modulo a prime p , and $\|f\|_\infty \leq H$. B) To show that when the algorithm terminates that the output is correct, namely a complete irreducible factorization of f in $\mathbb{Z}[x]$.

To attack these goals we must prove certain statements about the algorithm. We specified many of the parameters in the above algorithm, but made a note of where each is used. The parameters which can be altered by the user are $\ell := 1.5$, $Z_{\text{bound}} := 15$, $\delta := .99$, and $\eta := .51$, the rest of the parameters values must be derived by finding values which allow the theoretical analysis to hold we will highlight the inequalities we must be checked for alternate values. The parameters which we have specified are experimentally derived for their practical benefits.

The other parameters which are used in the algorithm and must be checked are: $B := r + 2$, $\text{NoVec}_{\text{bound}} := .937r - .0001$, and $E_{\text{bound}} := 1.6r^2$.

Throughout the algorithm there is a lattice basis given by the rows of the matrix M . The lattice given by M is changed throughout the algorithm (but never by LLL itself). To handle any changes to the lattice generated by M we introduce the Active Determinant.

Definition 1. *Given a lattice L and any basis b_1, \dots, b_s of L we define the Active Determinant (henceforth AD) to be the product of the norms of b_1^*, \dots, b_s^* the standard Gram-Schmidt (henceforth G-S) Orthogonalization of b_1, \dots, b_s . That is $\text{AD}(L) := \prod \|b_i^*\|$.*

The AD is an invariant of L , and in particular if L is given by the rows of M and L' is given by the rows of $\text{LLL}(M)$ then $\text{AD}(L) = \text{AD}(L')$ (and $L = L'$).

We will now use the AD to prove a bound on the number of rows of M throughout the algorithm, and a weak bound on the size of the G-S lengths of vectors which are removed in step 4(a)iv in Algorithm 2, later we will prove a more precise bound. We do this with a type of induction step, we will assume that s has yet to break the bound, then show that it would be impossible to break the bound a first time.

Lemma 1. *If s has yet to exceed $\lfloor (\beta = 1.21)r \rfloor$ then there is at most one vector with G-S norm $> 2^{(\text{NoVec}_{\text{bound}} + .0001 = .937r)}$.*

To show this Lemma is quite complex and we will give several sub-lemmas before finally having the proof. If there are s vectors which are α -reduced and the final vector has G-S length $\leq \sqrt{B}$ then each vector has actual Euclidean norm $\leq (\alpha)^{(s-1)/2} B^{1/2}$ (from LLL's paper or Grad feeding paper for simple proof if needed).

Now we want to give a very tight estimate of the bit-length of this bound. In an attempt to be as general as possible I propose the following: let $2^{\gamma r}$ be the trial bound for some constant value of γ , let βr be some potential bound on the maximum value of s (1.21 for the specific values we have chosen), α (in our case = 1.334) the LLL-quality bound as above (typically $1/(\delta - \eta^2)$), Z_{bound} the minimum value of r for which this algorithm gets called (in our case 15), and $B(r)$ the accuracy bound on the length of any target vector (more on this later, for now we use $B(r) = r + 2$). To state these 'dual' bounds I'll give the generic variable and a substituted value in parentheses, for instance $(\gamma = .374)r$ represents the quantity γr and that for our suggested inputs the recommended value of γ is .374.

Lemma 2. *If the value of s has remained $\leq (\beta = 1.21)r$ thus far in the algorithm then just after step 4(a)iv in Algorithm 2 each row of M has Euclidean norm (and G-S norm) bounded from above by $2^{(\gamma = .374)r}$.*

Calling this a lemma is a bit odd, it's more of a condition to be checked for a given set of parameters.

For checking the viability of a set of parameters (including the validity of some value for γ) make sure the following equation holds for the specific constants:

$$(\gamma - (\beta/2) \log_2(\alpha)) \cdot Z_{\text{bound}} \geq 1/2(\log_2(B(Z_{\text{bound}})) - \log_2(\alpha))$$

AND the following equation for all values of $r \geq Z_{\text{bound}}$

$$(\gamma - (\beta/2) \log_2(\alpha)) \geq (1/2) \left(\frac{B'(r)}{B(r) \ln(2)} \right)$$

The first equation shows that $2^{\gamma r}$ is an appropriate upper bound of the Euclidean norms at the smallest possible value of r and the second shows that if the first equation holds for the smallest value of r then the LHS grows faster than the RHS for all values of r so it must continue to be an upper bound for all other values of r . Both equations hold for $\gamma = .374$, $\beta = 1.21$, $Z_{\text{bound}} = 15$, $\alpha = 1.334$, and $B(r) = r + 2$, thus after step 4(a)iv in Algorithm 2 we know all rows in M have norm $\leq 2^{.374r}$, provided $s \leq 1.21r$.

Now, once a γ has been found we must check that $\gamma \ll \ell$ if some input values allow only values of γ larger than or even very near ℓ then the proofs will not work for those input values. In general these inequalities get more lenient as ℓ increase, Z_{bound} increases, and α decreases. We now pick $\text{NoVec}_{\text{bound}}$ as just under the average of ℓ and γ . If new vectors are added to an already reduced lattice with G-S norm of just over ℓr bits and if all previous vectors had G-S norm bounded by γr bits then there can only be one basis vector of G-S norm larger than the average of ℓr and γr (as LLL switches preserve the products of the G-S norms of the switched vectors and never increase the maximum G-S norm of the switched vectors). We should choose the number of bits we add in the the No Vector case to respect this bound as well. Thus set $\text{NoVec}_{\text{bound}}$ to $(\ell + \gamma)/2 - \epsilon$ for some small, but not too small, ϵ (the need for epsilon is to prove that, just after adjoining a new column in step 11a of Algorithm 3 in the No Vector Case we must show that the Euclidean Norm of this vector is no larger than $(\ell + \gamma)/2$ while it's actual size is the square root of the old vector squared plus the new entry squared).

This is also the first place when the rounding in step 7 of Algorithm 3 has some impact. In particular we set the scaling so that the largest new entry (if exact) would be $\geq 2^{(\ell+\gamma)/2-\epsilon-1}$ and $< 2^{(\ell+\gamma)/2-\epsilon}$, however the entry we actually add is not exact. We now give bounds for the difference between $\tilde{\mathbf{y}} := U \cdot \tilde{\mathbf{x}}$ and $\mathbf{y}/2^k := U \cdot \mathbf{x}/2^k$.

Lemma 3. *The largest possible error created by the rounding of step 7 in Algorithm 3 is $\leq \frac{\sqrt{r}}{2^{(1-\gamma)r}}$, that is $\|\tilde{\mathbf{y}} - \mathbf{y}/2^k\|_{\infty} \leq \frac{\sqrt{r}}{2^{(1-\gamma)r}}$.*

This lemma follows from a bound on the size of each row in U and and bound on the rounding error of each element in \mathbf{x} . We have, by construction, that: $\tilde{\mathbf{y}} - \mathbf{y}/2^k = U \cdot (\tilde{\mathbf{x}} - \mathbf{x}/2^k)$. The i^{th} entry in $\tilde{\mathbf{y}} - \mathbf{y}/2^k$ is the inner product of the i^{th} row of U and $(\tilde{\mathbf{x}} - \mathbf{x}/2^k)$. Each row of U is a sub-vector of one the b_i thus has Euclidean norm $\leq 2^{\gamma r}$. If we let the i^{th} row of U be denoted $U[i]$ then the i^{th} entry of our target vector is $U[i] \cdot (\tilde{\mathbf{x}} - \mathbf{x}/2^k)$ which equals $\|U[i]\| \cdot \|(\tilde{\mathbf{x}} - \mathbf{x}/2^k)\| \cdot \cos \theta$ for some value of θ . Thus this entry is bounded by $2^{\gamma r} \cdot \|(\tilde{\mathbf{x}} - \mathbf{x}/2^k)\|$, but each entry in $(\tilde{\mathbf{x}} - \mathbf{x}/2^k)$ is bounded by $1/2^r$ proving the lemma.

To confirm that a custom ϵ is large enough confirm that $\gamma < \ell$ and that the following equation holds for the given values of γ , ℓ , Z_{bound} , and ϵ :

FIXME: The rounding errors come into play here. See pen and paper.

$$2^{(\gamma-\ell)Z_{\text{bound}}} + 2^{-2\epsilon} \leq 1$$

(The derivative of both sides shows that so long as this holds and $\gamma < \ell$ then it will hold for all values of $r \geq Z_{\text{bound}}$.) In our case, with $\gamma = .374$, $\ell = 1.5$, Z_{bound} , the equation holds for very small values of ϵ in the algorithm above we put .0001 but values between .1 and .00001 suffices just as well. So we set $\text{NoVec}_{\text{bound}}$ to $.937r - .0001$ and now re-offer that targeted lemma:

Lemma 4. *If the value of s has remained $\leq (\beta = 1.21)r$ thus far in the algorithm then there is at most one row of M with G-S norm $> 2^{((\ell+\gamma)/2-.937)r}$.*

The logic was given in the preceding description.

Now we are ready to return to the discussion of Active Determinant AD and how one could prove that s would never exceed βr .

The next two lemmas give an upper bound for AD and a lower bound for AD, respectively, which should remain true at every step in the algorithm.

Lemma 5. *Just after step 4(a)iv in Algorithm 2 the AD is bounded from above by $B^{(s/2)}(\alpha)^{(s^2/4-s/4)}$ where s is the number of rows of M .*

This lemma follows simply from what it means to be α -reduced, namely $\|b_i^*\|^2 \leq \alpha \|b_{i+1}^*\|^2$ and the fact that after the removal step we know that $\|b_s^*\|^2 \leq B$ (or perhaps replace the above B by $B \cdot (1 + \epsilon)$ for some $0 \leq \epsilon \leq 1$ determined by the quality of our G-S data).

The next lemma bounds the AD from below:

Lemma 6. *Provided that s has yet to exceed $(1.21 = \beta)r$, then at any moment in the algorithm the AD is bounded from below by $2^{(s-r)\ell r}$ where s is the number of rows of M .*

Proof. There are 4 steps which alter M in any way. Step 1 in Algorithm 2 initializes M as an $r \times r$ identity matrix, which has $s = r$ and $\text{AD} = 1$.

Step 11a in Algorithm 3 has two possible outcomes. If a vector is added of the form $(0, \dots, 0, \lfloor p^a/2^k \rfloor)$ then s is increased by 1 and AD is multiplied by a factor $\lfloor p^a/2^k \rfloor$ which is chosen to be larger than $2^{(\ell r)}$ (the other G-S vectors are only augmented by a zero entry). In the case that no vector is added s is unaltered while the G-S vectors each have an entry augmented which cannot decrease their norm. So if the lemma holds prior to this step it must hold after this step.

Step 4(a)ii in Algorithm 2 calls LLL on M for which AD is invariant.

Finally, in step 4(a)iv in Algorithm 2 s is decreased by 1 for each removed vector. As each removed vector is the final vector it cannot impact the G-S norms of any remaining vectors. Thanks to lemma 1 we know that each removal but one decreases AD by a factor $\leq 2^{r \cdot (\ell + \gamma)/2}$. To handle the one potentially large vector (which will later be called a 'bad vector') we note that if s increased by 1 just prior to the LLL step and now a large vector is to be removed then the new vector was had the maximum G-S norm of any vector at the time it was attached to M (all others had G-S length $\leq 2^{\gamma r}$), and LLL never increases the maximum G-S length, and there is only vector at a time of G-S larger than $2^{r(\ell + \gamma)/2}$ so this large vector cannot have a net impact of decreasing AD (it's net impact was to leave s fixed while multiplying AD by a factor ≥ 1) (it went out with a G-S norm lower to or equal to the G-S norm when it came in).

Finally we can show that s will never exceed βr by taking the first time when s might break the bound and showing that the above two lemmas would lead to a contradiction. We want to show that the lower bound for AD must exceed the upper bound of AD for a given value of s . The logic is this: s only ever increases by one, when we add a vector, then after the removal step (step 4(a)iv of Algorithm 2) we get the upper bound for AD again and can only add a vector by returning to the add vector step (step 11a of Algorithm 3), so for s to grow beyond the value n then there must be a removal step which terminates with $s = n$, it is at that point when both bounds must apply and contradict each other.

Lemma 7. *Throughout the algorithm s will never exceed a value of $(1.21 = \beta)r$.*

The algorithm begins with $s = r$ so the above lemmas all hold until the first time when s might exceed βr . We will provide the needed inequalities with the format $s = \beta r$, since s can only take integer values these inequalities should be checked for a small range of values for β . By

setting the lower bound of AD above the upper bound of AD and using logarithms we want to check the following inequality for values of β , ℓ , α , Z_{bound} , and $B(r)$:

$$[(\beta - 1)\ell - \beta^2 \log_2(\alpha)/4] \cdot Z_{\text{bound}} + \beta \log_2(\alpha)/4 > \beta \log_2(B(Z_{\text{bound}}))/2$$

Also we should confirm the following for all values of $r \geq Z_{\text{bound}}$:

$$[2(\frac{\beta - 1}{\beta})\ell \ln(2) - \beta \ln(\alpha)/2] > \frac{B'(r)}{B(r)}$$

Now that we have bounded the number of rows in M , the AD, the size of reduced G-S lengths, and the size of removed vectors we can bound the number of calls to LLL.

We do this by classifying the calls to LLL into three cases. Before each call to LLL a new column is added to M and in some cases a new row. Specifically each call to LLL was preceded by a successful call to Algorithm 3 and is followed by a vector removal step (step 4(a)iv in Algorithm 2). We use this grouping to classify each call as a 'No Vector Case', a 'Good Vector Case', or a 'Bad Vector Case'. The 'Good Vector Case' is the most standard case, we consider a call to LLL part of the 'Good Vector Case' when a vector is added to M by Algorithm 3 and at the time just prior to removal (in step 4(a)iv in Algorithm 2) all vectors have G-S norms $\leq 2^{r \cdot (\ell + \gamma)/2}$ (note that it is not necessary to actually compute which case we are in, the case distinction is only of theoretical importance). The 'Bad Vector Case' is when a vector is added to M by Algorithm 3 and at the time just prior to removal there is a vector with G-S norm $> 2^{r \cdot (\ell + \gamma)/2}$. Finally the 'No Vector Case' is when a column is adjoined to M by Algorithm 3 but not an additional vector (as decided in step 9 of Algorithm 3). If we let n_{gv} , n_{bv} , and n_{nv} be counters for the number of times each case has taken place so far, then at any given point in the algorithm the number of calls to LLL so far is $n_{\text{gv}} + n_{\text{bv}} + n_{\text{nv}}$. We will also use the counter n_{rv} for the number of vectors which are removed and have G-S length $\leq 2^{r \cdot (\ell + \gamma)/2}$ at the moment of their removal.

We now must show how each case impacts the AD. We begin by recalling the impact of adjoining a new vector.

Lemma 8. *Each time a vector is adjoined in step 11a of 3 the AD is increased by a factor $\geq 2^{\ell r}$.*

We mentioned this earlier, here is some more detail. At the moment of Algorithm 3 both Lemma 5 and 2 apply to M as nothing alters M after the removal and before Algorithm 3. When M finally is altered it is in step 11a and the old M becomes $\begin{bmatrix} \mathbf{0} & \tilde{P} \\ M & \tilde{\mathbf{y}}_j \end{bmatrix}$. If b_1^*, \dots, b_s^* was the GSO just before step 11a and we let $\hat{b}_0^*, \dots, \hat{b}_s^*$ be the GSO with shifted indices just after step 11a then it is clear to see that $\hat{b}_i^* = (b_i^* | 0)$ since \hat{b}_0^* has 0 entries in each column but the final column then \hat{b}_i modulo \hat{b}_0 over \mathbb{R} simply zeros the final column to give $(b_i | 0)$. Thus the product of G-S norms, AD, after augmenting is a factor \tilde{P} times the AD before augmenting. However we know that $2^{\ell r} \leq \tilde{P}$ by the choice of k and rounding in steps 6 and 7 respectively.

Also we'll need the impact of adjoining a column without a vector.

Lemma 9. *In the No Vector Case when a column is adjoined in step 11a of 3 the AD is increased by a factor $\geq 2^{r \cdot (\ell - \gamma)/2 - \epsilon - 1 - \epsilon'}$.*

In the No Vector Case we adjoin an entry to each row vector, suppose that the largest such entry is adjoined to the k^{th} row. Let's say that this row had length L just prior to augmenting and that the new entry has size E then this row had length L and now has length $\sqrt{L^2 + E^2}$. The AD is independent of the order of the rows, so to see that the AD must increase we will

(only in the this theoretical argument) rearrange the vectors so that the k^{th} row is now the first row. Adding an entry to each row can only increase their G-S lengths (no matter what ordering we have), and the first vector's Euclidean length is its G-S length. Thus the AD has increased by a factor of $\sqrt{L^2 + E^2}/L$. Now we concentrate on finding a lower bound for this quantity.

First we work with the square $\frac{L^2 + E^2}{L^2} = 1 + \frac{E^2}{L^2}$. We have an upper bound on $L \leq 2^{\gamma r}$, which is in the denominator so $1 + \frac{E^2}{L^2} \geq 1 + \frac{E^2}{2^{2\gamma r}}$. The selection of the new entry included a constraint on its absolute value, but the entry is then rounded. So we know that $E > 2^{(\ell+\gamma)/2-\epsilon-1} - \frac{\sqrt{r}}{2^{(1-\gamma)r}}$. This now puts a lower bound of the amount which AD has been increased by at $\geq 1 + \frac{E^2}{2^{2\gamma r}} \geq 1 + (2^{(\ell-\gamma)r/2-\epsilon-1} - \sqrt{r}/2^r)^2$. Now I want to introduce a second epsilon, ϵ' such that this AD increase is $\geq 2^{r \cdot (\ell-\gamma)/2-\epsilon-1-\epsilon'}$ for simplicity we'll call this quantity (only in this section) $X/2^{\epsilon'}$ and let's name $\sqrt{r}/2^r$, the error term, as δ . So we know need to show that $X^2 - 2X\delta + \delta^2 + 1 - X^2/2^{2\epsilon'}$ is greater than or equal to zero. We can complete the square to get:

$$\left(\sqrt{\left(1 - \frac{1}{2^{2\epsilon'}}\right)}X - \frac{1}{\sqrt{\left(1 - \frac{1}{2^{2\epsilon'}}\right)}}\delta\right)^2 + 1 - \frac{1}{2^{2\epsilon'} - 1}\delta^2$$

Which is greater than zero if $2^{2\epsilon'} - 1 \geq \delta^2$. We can see that for the various input values we can handle $\epsilon' = .000001$, but simply check for a suggested value of ϵ' . (This argument is not used in any practical way, simply to lower bound the impact of a No Vector Case (so that we might later upper bound the number of times it could occur)).

Finally we need to show the impact of the 'Bad Vector Case', this is the case when the No Vector condition was not satisfied (so a vector is added to the lattice) and after LLL there is a vector of G-S length $> 2^{\gamma+\ell}/2$ in the final position. (Perhaps I should prove that LLL will place this large vector in the final spot?... FIXME)

Lemma 10. *In the Bad Vector Case when the large vector is removed in step 4(a)iv of 2 the AD is increased by a factor ≥ 2 . (Possibly more, 2^{3r} , up to an argument FIXME).*

In this situation we know that $B \cdot \sum_{i=1}^{s-1} (1 + \eta)^i \geq \frac{[\tilde{P}] - B}{\lceil \|\tilde{\mathbf{y}}_j\|_\infty \rceil}$ (note that this should be the real NoVec condition and we'll adjust it in the actual algorithm in just a bit FIXME).

We now CLAIM that this condition implies that there is a k such that, just prior to augmenting, we have $2 \|\mathbf{M}[k]\| \leq \|\tilde{\mathbf{y}}_j[k]\|$ (FIXME: thanks to tighter inequalities we can make this $2^{\nu r} \|\mathbf{M}[k]\| \leq \|\tilde{\mathbf{y}}_j[k]\|$ for some ν around .3). This is saying that one of the new entries is at least twice the size of the row it was attached too.

PROOF OF CLAIM HERE: To check the claim we observe that if it were not true we would have an upper bound on $\lceil \|\tilde{\mathbf{y}}_j\|_\infty \rceil$, namely $\lceil \|\tilde{\mathbf{y}}_j\|_\infty \rceil \leq 2 \cdot 2^{\gamma r} + \frac{\sqrt{r}}{2^{(1-\gamma)r}}$. Now we can arrive at a contradiction by showing that $B(r) \cdot \lceil \|\tilde{\mathbf{y}}_j\|_\infty \rceil \cdot ((1/\eta)(1 + \eta)^s - (1/\eta) - 1) + 1$ is strictly less than $[\tilde{P}]$. We have a lower bound on P and an upper bound on the other monster.

CHECK EQUATION:

$$B(r) \cdot \left[(2 \cdot 2^{\gamma r} + \frac{\sqrt{r}}{2^{(1-\gamma)r}}) \cdot ((1/\eta)(1 + \eta)^s - (1/\eta) - 1) + 1 \right] < 2^{\ell r} - \frac{\sqrt{r}}{2^{(1-\gamma)r}}$$

Check this above equation for all r larger than the minimum and substitute the inputs $(\gamma, \eta, B(r), \ell)$ this one is not very tight so there is room for slop.

END OF PROOF OF CLAIM.

We'll use $\Lambda(A)$ to denote the lattice generated by the integer combinations of the rows of A .

Suppose \hat{M} is the output of LLL just prior to step 4(a)iv in Algorithm 2, and $\hat{M}[1, \dots, -2]$ is the output of LLL but dropping the final row. We will let $\pi_{[1, \dots, -2]}(A)$ denote the projection of some integer matrix A which drops the final element of each row. If M is the matrix before being augmented in step 11a of 3 then we can make two cases. Case 1: $\Lambda(M)$ does not equal $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$, Case 2: $\Lambda(M)$ equals $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$.

Case 1: $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}))$ equals $\Lambda(M)$, so in this case $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$ is a sub-lattice of $\Lambda(M)$, and the index of $[\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2])) : \Lambda(M)]$ is an integer ≥ 2 meaning that the determinant of $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$ is at least two times the determinant of $\Lambda(M)$ and the inverse of $\pi_{[1, \dots, -2]}$ can only increase the determinant. So the net impact of removing the large final vector must be an increase of at least a factor 2 of the AD.

Case 2: We are interested in showing that the G-S length of $\hat{M}[-1]$ (the last row of the matrix after LLL) is no more than $\tilde{P}/2$, which would imply that its removal after its inclusion would have increased the AD by a factor greater than or equal to 2. The G-S length of $\hat{M}[-1]$ is the minimum value in $\{\|\hat{M}[-1] - \sum r_i \cdot \hat{M}[i]\| \mid \forall r_i \in \mathbb{R}\}$.

In case 2 we know that $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$ equals $\Lambda(M)$ which equals $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}))$, while $\Lambda(M)$ and $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$ have the same dimension (and the rows of M are linearly independent).

So there must be integers a_i such that $\hat{M}[-1] - \sum a_i \hat{M}[i] = \pm(0, \dots, 0, \tilde{P})$. Now the earlier claim gives some k with $2 \|\hat{M}[k]\| \leq |\tilde{\mathbf{y}}_j[k]|$, so the row vector $\hat{M}[k]$ is in $\Lambda(\pi_{[1, \dots, -2]}(\hat{M}[1, \dots, -2]))$ meaning that some integer combinations of the rows in $\hat{M}[1, \dots, -2]$ gives $(\hat{M}[k], S)$ (some S) which is equivalent to $(\hat{M}[k]|\tilde{\mathbf{y}}_j[k])$ modulo $(0, \dots, 0, \tilde{P})$. Now that means that in the set $\{\|\hat{M}[-1] - \sum r_i \cdot \hat{M}[i]\| \mid \forall r_i \in \mathbb{R}\}$ is $\pm(0, \dots, 0, \tilde{P}) \mp (\frac{\tilde{P}}{S}) \cdot (\hat{M}[k], S)$ which has norm $\frac{\|\hat{M}[k]\| \cdot \tilde{P}}{S}$ and the smallest value of S is $|\tilde{\mathbf{y}}_j[k]|$. Our earlier claim now implies that this combination has norm $\leq \tilde{P}/2$ and is an upper bound for the G-S length of $\hat{M}[-1]$. Proving the lemma for case 2.

Now we are ready to bound the total number of calls to LLL. Each call fits into one of the three cases Good, Bad, and No Vector. We also know that each vector which is removed outside of the Bad Vector Case has an upper bound on its G-S length of $2^{r \cdot (\ell - \gamma)/2}$. Each new vector in the Good case increases the AD by a factor $\geq 2^{\ell r}$, each No Vector case increases AD by $\geq 2^{r \cdot (\ell - \gamma)/2 - \epsilon - 1 - .001}$, and the Bad Vector case inserts and removes one vector with a net impact of increasing AD by a factor at least 2. These facts allow us to put upper and lower bound on the AD just after removal. Recall our earlier counters n_{rv} is the number of removed vectors so far, $n_{\text{gv}}, n_{\text{bv}}, n_{\text{nv}}$ are the number of times there has been a good vector LLL, bad vector LLL, and no vector LLL respectively.

Then just after removal we have the upper bound on AD from earlier of $B^{(s/2)}(\alpha)^{(s^2/4 - s/4)}$. Here s , the current number of vectors, is given by $n_{\text{gv}} - n_{\text{rv}} + r$ and is bounded from above by βr . We can also say that the AD is bounded from below by totalling the minimal impact of our actions so far that is: $\text{AD} \geq 2^{\ell r \cdot n_{\text{gv}} + n_{\text{bv}} + (r \cdot (\ell - \gamma)/2 - \epsilon - 1 - \epsilon') \cdot n_{\text{nv}} - r \cdot (\ell + \gamma)/2 \cdot (n_{\text{gv}} + r - 1)}$ (note that we've replaced n_{rv} by an upper bound $n_{\text{gv}} + r - 1$). To see this just note that AD is only changed by the marked events, removals, and augmentations, not LLL calls, and the AD begins with a value of 1, we have bounded the amount that AD can decrease after each removal and the number of removals, while ensuring that the impact of adjoining new vectors is larger than the size of these removed vectors. While the nature of our search guarantees that the vectors can not have AD too large. This means that we can bound the number of times each given step could happen before the problem must be solved.

We'll start by bounding the number of 'good vector' calls to LLL. In the worst case $n_{\text{bv}}, n_{\text{nv}}$ are 0 and $n_{\text{rv}} = n_{\text{gv}} + r - 1$.

By taking \log_2 of the above inequality and collecting we get the following inequality which must hold:

MAJOR CHECK EQUATION:

$$([\beta^2/4\log_2(\alpha) + (\ell + \gamma)/2] \cdot r + (\beta/2)\log_2(B(r)) - [\beta/4\log_2(\alpha) + (\ell + \gamma)/2]) \cdot \frac{2}{\ell - \gamma} > n_{\text{gv}}$$

With our standard set of inputs we get a linear upper bound for $n_{\text{gv}} \leq 2.103r$. (In this case I used $.1688r$ for the impact of the non- r terms.)

This also provides a bound on the number of removed vectors (recall that this does not include 'bad vectors'), $n_{\text{rv}} \leq 3.17r$.

For checking the 'Bad Vector Case' we run a similar analysis but this time we set $n_{\text{gv}}, n_{\text{nv}}$ to zero and then maximize $n_{\text{rv}} = r - 1$.

Collecting and such we get:

MAJOR CHECK EQUATION:

$$[\beta^2/4\log_2(\alpha) + (\ell + \gamma)/2] \cdot r^2 + (\beta/2)\log_2(B(r))r - r \cdot [\beta/4\log_2(\alpha) + (\ell + \gamma)/2] > n_{\text{bv}}$$

with our standard values we can bound the number of bad vector cases by $n_{\text{bv}} \leq 1.184r^2$.

Finally for the 'No Vector Case' we set $n_{\text{bv}}, n_{\text{gv}}$ to zero and $n_{\text{rv}} = r - 1$ and collect in the same way to arrive at:

MAJOR CHECK EQUATION:

$$([\beta^2/4\log_2(\alpha) + (\ell + \gamma)/2] \cdot r^2 + (\beta/2)\log_2(B(r))r - r \cdot [\beta/4\log_2(\alpha) + (\ell + \gamma)/2]) / [\frac{\ell - \gamma}{2} \cdot r - 1 - \epsilon - \epsilon'] > n_{\text{nv}}$$

Note that this one is the least optimal as it would be impossible to have $s = \beta r$ with no good or bad vectors and $r - 1$ removed vectors.

In this sub-optimal way we still can show $n_{\text{nv}} \leq 2.384r$ with our standard inputs. Note that not all three of these bounds can be reached simultaneously. Even so we certainly now know that the number of calls to LLL is bounded by $1.184r^2 + 2.384r + 2.17r$.

Now we are one step from the complete complexity analysis, the amortization of LLL costs via globally bounding the number of LLL switches.

To handle the number of LLL switches we introduce a metric, called Progress P , which is an adaptation of an Energy function. At any point in the algorithm we can track the current Progress P by observing the dimension of M , the G-S lengths of the rows of M , and the number of removed vectors and bad vector cases. Here is the metric, where $l_i := \log_{\frac{1}{\sqrt{\delta}}}(\|b_i^*\|)$ the GSO of the rows of M :

$$P := \beta r n_{\text{bv}} + \beta \frac{(\gamma + \ell)}{2} \log_{\frac{1}{\sqrt{\delta}}}(2) r^2 n_{\text{rv}} + \begin{cases} \sum_{i=1}^s (i \cdot l_i) & l_i \leq \frac{(\gamma + \ell)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2) \quad \forall i \\ \sum_{i=1}^{j-1} (i \cdot l_i) + (j - 1) + \sum_{i=j+1}^s ((i - 1) \cdot l_i) & l_j > \frac{(\gamma + \ell)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2) \end{cases}$$

We showed earlier that at most one vector exceeds $\frac{(\ell + \gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$ at a time, so this metric will apply throughout the algorithm.

Now we need to show that this $P = \mathcal{O}(r^3)$, never decreases, and that the number of LLL-switches made throughout the algorithm is bounded by P . The analysis naturally breaks up into

what happens when vectors are removed, what happens when LLL operates, and what happens when we augment the lattice.

When augmenting we want to make sure that the progress P cannot decrease.

When removing vectors we want to make sure of the same.

When LLL operates we want to make sure that progress increases by 1 per switch (and doesn't decrease via any other actions).

Augmenting:

If we adjoin a row then the analysis is simple, r , n_{rv} , n_{bv} do not change, so we need to see the impact on the l_i . Let's say that l_1, \dots, l_s were the values just prior to augmenting and l'_1, \dots, l'_{s+1} are the values after. Then we have $l'_1 > \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$ and $l'_{i+1} = l_i$. So $\sum il_i$ is equal to $0 + \sum(i-1)l'_i$, and the progress is unaffected by augmenting a row.

If we do not augment a row then we adjoin a column. Again only the l_i are changed, with each G-S length possibly increasing but not decreasing. Also our earlier analysis showed that each G-S length is kept $\leq \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$. So in this case Progress might be increased by these new entries but not decreased.

Removing: The removing of vectors falls into two categories, the removed vector has $l_s > \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$, or it doesn't. When that removed vector has $l_s > \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$ then we increase the n_{bv} by 1, which replaces the $s-1$ term in Progress by a βr , and our earlier work shows that this does not decrease Progress ($\beta r \geq s-1$).

In the other, more standard, case we are removing a vector whose weight in Progress was either sl_s or $(s-1)l_s$ and incrementing n_{rv} whose weight is $(\beta r) \cdot (\frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2))$ which cannot decrease progress ($s-1 < s \leq \beta r$) and ($l_s \leq \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$).

So each removal can only increase the Progress P metric.

LLL operations:

There are two operations which alter M made by LLL. They are 1) swaps of two consecutive rows and 2) subtracting integer multiples of earlier rows from a given row. The second type of operation has no impact on G-S length and thus no impact on Progress. We assume that the LLL which is chosen for our application respects the input $1/\delta$ in the sense that every swap of the i^{th} and $(i+1)^{\text{th}}$ rows 'moves a factor of at least $1/\delta$ G-S length'. That is if l_i, l_{i+1} are just prior to a swap and l'_i, l'_{i+1} are just after the swap we have the following:

- a) $l_i \geq l_{i+1}$ otherwise a switch would not be made
- b) $l_{i+1} \leq l'_i \leq l_i - 1$ that is the first G-S length can't be made too small but is made smaller.
- c) $l_{i+1} + 1 \leq l'_{i+1} \leq l_i$ that is the second G-S length can't be made too large but is made larger.

d) and of course $l_i + l_{i+1} = l'_i + l'_{i+1}$ that is the product of the G-S lengths can't change.

Note also that none of the other l_j are altered in anyway by the swap.

These 4 equations are enough to show that each switch increases P by at least 1.

Keeping our analysis focused on a single switch of the i^{th} and $(i+1)^{\text{th}}$ rows we now observe the possible cases:

1) Both l_i and l_{i+1} began as $\leq \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$

2) l_i and l'_i are $> \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$

3) l_i and l'_{i+1} are $> \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$

4) l_i is $> \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$ and l'_i and l'_{i+1} are not.

Note that if $l_{i+1} > \frac{(\ell+\gamma)}{2} r \log_{\frac{1}{\sqrt{\delta}}}(2)$ then a switch wouldn't take place since $l_i < l_{i+1}$.

In all cases we will look only at the parts of Progress which are impacted by the swap.

Case 1) is the most standard: $il_i + (i + 1)l_{i+1}$ becomes $il'_i + (i + 1)l'_{i+1}$

So $il_i + (i + 1)l_{i+1} + 1 = i(l'_i + l'_{i+1} - l_{i+1}) + (i + 1)l'_{i+1}$ by eqn d) this equals $il'_i + il'_{i+1} + (l_{i+1} + 1)$ which is $\leq il'_i + (i + 1)l'_{i+1}$ by eqn c). This showed the the progress before plus one is less than equal to the progress after, so the switch increased progress by at least 1.

Case 2) We replace $(i - 1) + il_{i+1}$ by $(i - 1) + il'_{i+1}$. So $(i - 1) + il_{i+1} + 1 \leq (i - 1) + i(l_{i+1} + 1)$ since $i \geq 1$. This is now $\leq (i - 1) + il'_{i+1}$ by eqn c). Showing that the progress is at least 1 larger in this case.

Case 3) We replace $(i - 1) + il_{i+1}$ by $il'_i + i$. So $(i - 1) + il_{i+1} + 1 = i + il_{i+1}$ which is $\leq i + il'_i$ by eqn b). Showing that progress increased by at least 1 in this case.

Case 4) We replace $(i - 1) + il_{i+1} + \sum_{j=i+2}^s (j - 1)l_j$ by $il'_i + (i + 1)l'_{i+1} + \sum_{j=i+2}^s jl_j$. This is the first place we use the fact that the $l_j \geq 0$, to see this note that the G-S lengths begin at 1, augmentations cannot decrease them, and in LLL the minimum G-S length is never decreased thanks to eqn b). Since $l_j \geq 0$ for $j > i + 1$ we can ignore the change from $\sum (j - 1)l_j$ to $\sum (j)l_j$. So $(i - 1) + il_{i+1} + 1 \leq i + (i + 1)l_{i+1}$ since $i \geq 0$. Now this is $\leq i + (i + 1)l_{i+1} + (1 + il'_i)$ to get $(i + 1)(l_{i+1} + 1) + il'_i$ which is $\leq il'_i + (i + 1)(l'_{i+1})$ showing that in this case progress increases by at least 1 (really 2 or more).

Now we've shown that Progress never decreases and increases by 1 for every switch made by LLL. Finally we plug in the largest possible values for the various parameters into P to get an upper bound for the number of switches. The best time to check this is just after removal since the l_i will all be less then $\gamma r \log_{\frac{1}{\sqrt{8}}}(2)$.

At this point we'll use all of our specific inputs, $\beta = 1.21$, $n_{bv} \leq 1.184r^2$, $(\ell + \gamma)/2 = .937$, $n_{rv} \leq 3.17r$, $\log_{\frac{1}{\sqrt{8}}}(2) \leq 138$. This gives $P \leq 1.433r^3 + 496r^3 + 37.78r^3 + 31.23r^2 = \mathcal{O}(r^3)$!

Accuracy To ensure the accuracy we need to ensure that the lattice spanned by the first r entries of the rows of M contains all target vectors, we'll use $\pi(M)$ to represent the first r entries of the rows of M . In the original van Hoeij paper a proof is given that this condition along with a trial division is sufficient. More specifically if A) the $\text{rref}(\pi(M))$ is a 0-1 basis with each column containing exactly one non-zero entry of value 1, and B) you have a factorization of f then you have the complete irreducible factorization thanks to the uniqueness of the complete factorization, the uniqueness of the rref , and the fact that $\Lambda(\pi(M))$ contains all target vectors. (FIXME flesh this out don't just refer.)

Since the conditions A) and B) are checked by procedure Algorithm 6 we need only confirm $\Lambda(\pi(M))$ contains all target vectors. The algorithm begins with $\pi(M) = I_{r \times r}$ so $\Lambda(\pi(M)) = \mathbb{Z}^r$ which contains all 0-1 vectors, and thus all target vectors. The lattice $\Lambda(\pi(M))$ is only ever altered by the removal step (step 4(a)iv of Algorithm 2), indeed the other changes to M augment a new row (which has a 0 vector for the first r entries), augment a new column (which has no impact on $\pi(M)$), or apply unimodular operations to the rows of M (during LLL) which don't impact $\Lambda(\pi(M))$. To show that target vectors remain in $\Lambda(\pi(M))$ during removal we rely on the following fact about lattices and G-S lengths:

Fact 1 *Let a lattice L have basis b_1, \dots, b_s which has a GSO or b_1^*, \dots, b_s^* . For every $v \in L$ with $\|v\|^2 \leq B$ (for any real value of B), if $\|b_s^*\|^2 > B$ then $v \in \text{SPAN}_{\mathbb{Z}}\{b_1, \dots, b_{s-1}\}$.*

This means that the sub-lattice spanned by all but the final vector of a basis will contain all vectors of length \leq the G-S length of the final vector. The removal step only removes vectors in the final position with squared G-S length provably $> B(r)$ (which we are using at $r + 2$), thus the remaining lattice must contain all of the target vectors if each target vector has squared length $\leq B(r)$ (at each execution of step 4(a)iv of Algorithm 2).

Now we explore bounds on the squared norms of the target vectors. Let's suppose that the matrix has been augmented k times so far and in augmentation i we had a p -adic accuracy of a_i and augmented the c_i^{th} CLD after rounding, let's denote the augmented data as p^{a_i} and \mathbf{x}_{j_i} . I want to claim the following:

Lemma 11. *Let \mathbf{w}_g be a target 0-1 vector corresponding to the irreducible factor g of f over \mathbb{Z} (via the correspondence $g \equiv \prod f_i^{\mathbf{w}_g^{[i]}} \pmod{p^a}$). Now suppose that \mathbf{x}_j and p^a , the precision, are any CLD data which survive until at least step 6 of Algorithm 3 then in the standard case:*

$$|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j \rangle \pmod{\tilde{P}}| \leq \frac{2r}{2^r} + \frac{1}{\sqrt{E_{\text{bound}}} \cdot B(r)}$$

and if $\langle \mathbf{w}_g, \mathbf{x}_j \rangle = \text{CLD}_j(g)$ then:

$$|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j \rangle| \leq \frac{2r}{2^r} + \frac{1}{\sqrt{E_{\text{bound}}} \cdot B(r)}$$

Let's start with the following: $\langle \mathbf{w}_g, \mathbf{x}_j \rangle \pmod{p^a} = \text{CLD}_j(g)$.

Thus there exists a n such that $\langle \mathbf{w}_g, \mathbf{x}_j \rangle + n \cdot p^a = \text{CLD}_j(g)$, and thus $\langle \mathbf{w}_g, \mathbf{x}_j / 2^k \rangle + n \cdot p^a / 2^k = \text{CLD}_j(g) / 2^k$. Also we have $2^k \geq X_j \cdot B(r) \cdot \sqrt{1.6r^2}$ in both the No Vector case and the standard case. This means that $|\langle \mathbf{w}_g, \mathbf{x}_j / 2^k \rangle + n \cdot p^a / 2^k| \leq \frac{1}{\sqrt{E_{\text{bound}}} \cdot B(r)}$ by definition of X_j and the fact that g is a true factor of f . We also know that $|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j - \mathbf{x}_j / 2^k \rangle| \leq r / 2^r$ by the precision of rounding (with the minimum of 15 this is $\leq .0004$). Now we can do the following $|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j - \mathbf{x}_j / 2^k \rangle + n \cdot (p^a / 2^k - p^a / 2^k)| \leq r / 2^r + n / 2^r$ again by the precision of the rounding. Which gives $|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j \rangle + n \cdot [p^a / 2^k] - (\langle \mathbf{w}_g, \mathbf{x}_j / 2^k \rangle + n \cdot p^a / 2^k)| \leq r / 2^r + n / 2^r$ using the triangle inequality we can arrive at $|\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j \rangle + n \cdot [p^a / 2^k]| \leq (r + n) / 2^r + \frac{1}{\sqrt{E_{\text{bound}}} \cdot B(r)}$. To bound n we can observe that $\langle \mathbf{w}_g, \mathbf{x}_j \rangle$ is at most a sum of r numbers each reduced $\pmod{p^a}$ so $n \leq r - 1$ and in our case this means that $\langle \mathbf{w}_g, \tilde{\mathbf{x}}_j \rangle + n \cdot [p^a / 2^k]$ is already reduced $\pmod{[p^a / 2^k]}$ (because with $r > 15$ this is so small that another subtraction or addition of the modulus would make it larger). Giving the Lemma. The second statement is true if $n = 0$.

Theorem 1 *Just after the n^{th} successful call to Algorithm 3, for $n \leq E_{\text{bound}}$, we have $(\mathbf{w}_g, c_1, \dots, c_n) \in \Lambda(M)$. Here we let c_i come from the column corresponding with the i^{th} successful call, during which we used \mathbf{x}_{j_i} as the raw CLD data, 2^{k_i} as the scaling factor, and p^{a_i} as the p -adic precision during that successful call then we define $c_i := \langle \mathbf{w}_g, \tilde{\mathbf{x}}_{j_i} \rangle \pmod{\tilde{P}}$ (where \tilde{r} represents the rounding, and $P := p^{a_i}$). Further this vector $\|(\mathbf{w}_g, c_1, \dots, c_n)\|^2 \leq r + n \cdot [\frac{1}{E_{\text{bound}}} + \frac{4r}{2^r \sqrt{E_{\text{bound}}}} + \frac{4r^2}{2^{2r}}]$ which is $\leq B(r)(= r + 2)$.*

We proceed with induction. Assume that the theorem holds for $n - 1$ (as it does for $n = 0$ since the \mathbf{w}_g has largest possible squared norm when it has r entries each equal to 1 and $\Lambda(I) = \mathbb{Z}^r$), and we will show that it holds for all $n \leq E_{\text{bound}}$. Given that $(\mathbf{w}_g, c_1, \dots, c_{n-1}) \in \Lambda(M)$ just after the previous augmentation and that $\|(\mathbf{w}_g, c_1, \dots, c_{n-1})\|^2 \leq r + (n-1) \cdot [\frac{1}{E_{\text{bound}}} + \frac{4r}{2^r \sqrt{E_{\text{bound}}}} + \frac{4r^2}{2^{2r}}]$ which is $\leq B(r)(= r + 2)$ implies that $(\mathbf{w}_g, c_1, \dots, c_{n-1}) \in \Lambda(M)$ just after the removal in step 4(a)iv of Algorithm 2 thanks to the earlier fact. Now we want to ensure that $(\mathbf{w}_g, c_1, \dots, c_{n-1}, c_n)$ is in $\Lambda(M)$ just after augmentation. In the standard case we augment with the row vector $(0, \dots, 0, \tilde{P})$ and in both cases the column $U \cdot \mathbf{x}_{j_n}$ is augmented. Take whichever combination of rows in the old M form $(\mathbf{w}_g, c_1, \dots, c_{n-1}, c_n)$ and apply them to M augmented only by the new column to arrive at $(\mathbf{w}_g, c_1, \dots, c_{n-1}, \langle \mathbf{w}_g, \tilde{\mathbf{x}}_{j_n} \rangle)$, now we can subtract the newly augmented row as often as we need to create $(\mathbf{w}_g, c_1, \dots, c_n)$. This shows that, when the new row is added (not the No Vector

case) we must have the vector in the lattice $\Lambda(M)$. Now we argue that this vector $(\mathbf{w}_g, c_1, \dots, c_n)$ has the required norm, by merely applying the previous Lemma (squared) to get the increase, and using the specific $r > 15$ and $n \leq E_{\text{bound}}$ to show that the total is increased by just the one entry and the total remains $\leq r + 2$. This proves the theorem in the standard case.

Now we will argue that, in the No Vector Case, if this new row $(0, \dots, 0, \tilde{P})$ were in the final position it would have squared G-S length $> r + 2$ in which case our target vector would not need the $(0, \dots, 0, \tilde{P})$ vector to remain in the lattice (so it can be excluded from the start).

Imagine now that the new row is in final position. We'll let M be the matrix prior to any augmenting, then the G-S length in question would be the squared norm is the following vector: $(0, \dots, 0, \tilde{P}) - \sum m_i \cdot (M[i], \langle U[i], \lfloor \frac{\mathbf{x}_{jn}}{2^{kn}} \rfloor \rangle)$ for some specific real numbers m_i (in fact the G-S vector is the vector of minimal length over all reals m_i). We now look at two cases A) $\sum |m_i| < \frac{\lfloor \frac{p^{an}}{2^{kn}} \rfloor - \sqrt{B(r)}}{\lfloor \frac{\|U \cdot \mathbf{x}_{jn}\|_\infty}{2^{kn}} \rfloor}$ and the case B) $\sum |m_i| \geq \frac{\lfloor \frac{p^{an}}{2^{kn}} \rfloor - \sqrt{B(r)}}{\lfloor \frac{\|U \cdot \mathbf{x}_{jn}\|_\infty}{2^{kn}} \rfloor}$

In case A) we can rearrange the inequality to give:

$\sqrt{B(r)} < \lfloor \frac{p^{an}}{2^{kn}} \rfloor - (\sum |m_i|) \cdot \lfloor \frac{\|U \cdot \mathbf{x}_{jn}\|_\infty}{2^{kn}} \rfloor$ this is certainly $\leq \lfloor \frac{p^{an}}{2^{kn}} \rfloor - \sum (m_i \cdot \langle U[i], \lfloor \frac{\mathbf{x}_{jn}}{2^{kn}} \rfloor \rangle)$ but this is the last entry in the G-S vector in question so certainly it's squared length exceeds $B(r)$.

In case B) we need to investigate the rows of M (we still are using M just prior to the current augmentation). We let $M^*[i]$ denote the i^{th} row in the GSO of M . Then $M[i] = M^*[i] + \sum_{j < i} \mu_{i,j} M^*[j]$ and since M was LLL reduced we know that $|m_{i,j}| \leq \eta$. The vector we are interested in is $\sum_{i=1}^s m_i \cdot M[i] = \sum_{i=1}^s m_i \cdot (M^*[i] + \sum_{j < i} \mu_{i,j} M^*[j]) = \sum_{i=1}^s (m_i + \sum_{l > i} \mu_{l,i} m_l) \cdot M^*[i]$. Note that the $M^*[i]$ are pair-wise orthogonal and each has length ≥ 1 so $\| \sum_{i=1}^s m_i M[i] \|^2 \geq \sum_{i=1}^s (m_i + \sum_{l > i} \mu_{l,i} m_l)^2$ Thus is if there was just a single index i with $|(m_i + \sum_{l > i} \mu_{l,i} m_l)| > \sqrt{B(r)}$ then we are done. So now assume that no such i exists. Then $|m_s| \leq \sqrt{B(r)}$ and $|m_{s-1} + \mu_{s,s-1} m_s| \leq \sqrt{B(r)}$ so $|m_{s-1}| \leq \sqrt{B(r)}(1 + \eta)$. We will argue inductively that $|m_{s-i}| \leq \sqrt{B(r)}(1 + \eta)^i$. Assume this holds for $i < K$, we already know that $|m_{s-K} + \mu_{s-(K-1), s-K} m_{s-(K-1)} + \dots + \mu_{s,s-K} m_s| \leq \sqrt{B(r)}$ gives us $|m_{s-K}| \leq \eta \sqrt{B(r)}(1 + \eta)^{K-1} + \dots + \eta \sqrt{B(r)}(1 + \eta) + \eta \sqrt{B(r)} + \sqrt{B(r)} = \sqrt{B(r)}(1 + \eta)^K$.

Now we know that $\sum_{i=1}^s |m_i| \leq \sqrt{B(r)}(\sum_{i=0}^{s-1} (1 + \eta)^i)$ but this contradicts the condition that triggers the No Vector Case, since $\sqrt{B(r)}(\sum_{i=0}^{s-1} (1 + \eta)^i) < \frac{\lfloor \frac{p^{an}}{2^{kn}} \rfloor - \sqrt{B(r)}}{\lfloor \frac{\|U \cdot \mathbf{x}_{jn}\|_\infty}{2^{kn}} \rfloor} \leq \sum_{i=1}^s |m_i|$. Thus that final G-S length squared is $> B(r)$. This means that $\langle \mathbf{w}_g, \mathbf{x}_j \rangle = \text{CLD}_j(g)$ since no reduction needed to be done (the target vector exists in the lattice without $(0, \dots, 0, \tilde{P})$, and the previous lemma applies now to the No Vector Case as well (for the norm requirements). Giving the theorem.

Now we recall that $n < E_{\text{bound}}$ as proved in the previous section, so that if the algorithm ever terminates the output is accurate (a complete irreducible factorization of f).

First we must handle the termination of the algorithm before the overall complexity.

Termination

We've seen already that there is an upper bound on the number of LLL calls. Since there is no guarantee that LLL will be called at a given level of p -adic accuracy (other than the first one), we now will argue that there is some precision p^A for which LLL must be called or the problem must be solved (that is $\text{rref}(\pi(M))$ gives a solution). If this precision is reached one uses all coefficients repeatedly until another LLL call takes place rather than continue Hensel lifting. We do note that this level of Hensel lifting has never been observed and is in line with the amount of Hensel lifting as the previous best complexity bounds for factoring. This precision $A = \mathcal{O}(N(N + H))$ where N is the degree of f and H is $\|f\|_\infty$.

We rely on the basic method of BHKS. Also let W be the lattice spanned by the 0–1 target vectors in $\{0, 1\}^r$, then we can refer to $\Lambda(\pi(M)) = W$ or not, and we know that $W \subseteq \Lambda(\pi(M))$ by the previous section.

Lemma 12. *We want to show that if Algorithm 3 returns `False` for all CLDs at a accuracy p^A then $\Lambda(\pi(M))$ is in fact equal to W .*

Assume that Algorithm 3 fails for all CLDs and $\Lambda(\pi(M)) \neq W$. Let us define a vector for each true factor g_1, \dots, g_K by $\mathbf{w}_i := (e_{1,i}, \dots, e_{r,i}, \text{CLD}_0(g_i), \dots, \text{CLD}_{N-1}(g_i))$ where $g_i = \prod f_j^{e_{i,j}}$ over the p -adics (recall f_j are the r local factors of f). Then by Corollary 4.2 in BHKS we have $\|\mathbf{w}_i\| \leq \sqrt{r + (2^{N-1}N \|f\|_2)^2} =: B'$ (we'll call this quantity B').

We know that each CLD is bounded by $2^{N-1}N \|f\|_2$.

Now, since $W \subset \Lambda(\pi(M))$ and $W \neq \Lambda(\pi(M))$ then there is an i with $\pi(M[i]) \notin W$, let's call $\mathbf{v} := \pi(M[i])$ this vector not in W . Now, if \mathbf{x}_j represents the j^{th} CLD at precision p^A then let $\tilde{\mathbf{v}} := (\mathbf{v}[1], \dots, \mathbf{v}[r], \langle \mathbf{v}, \mathbf{x}_0 \rangle \bmod p^A, \dots, \langle \mathbf{v}, \mathbf{x}_{N-1} \rangle \bmod p^A)$. If $\|\tilde{\mathbf{v}}\|^2 > (r + N) \cdot [2^{N-1}N \|f\|_2 \cdot \sqrt{E_{\text{bound}}} \cdot 2^{(\ell+\gamma)r/2} \cdot B(r)]^2$ then Algorithm 3 will succeed, this takes the (rather weak) bound from Cor. 4.2 in BHKS, and the conditions in step 5 (also note that $\|\mathbf{v}\| \leq 2^{\gamma r}$). Let's name the square root of this quantity $K' := \sqrt{(r + N) \cdot [2^{N-1}N \|f\|_2 \cdot \sqrt{E_{\text{bound}}} \cdot 2^{(\ell+\gamma)r/2} \cdot B(r)]}$.

Now we follow the proof of Theorem 4.3 in BHKS. Mentioning only the parts we need. Adjust $\tilde{\mathbf{v}}$ so that it satisfies the conditions of Lemma 3.2 in BHKS which produces $\mathbf{b} = \tilde{\mathbf{v}} + \sum_{j=1}^K n_j \mathbf{w}_j$, here $n_j \in \{0, 1\}$ for all but one j and $n_j \in \{0 - e, 1 - e\}$ for that j where $e \in \{\mathbf{v}[1], \dots, \mathbf{v}[r]\}$, see BHKS for details on this. So we know that $|e| \leq 2^{\gamma r}$. Thus, if $\|\tilde{\mathbf{v}}\| \leq K'$ then $\|\mathbf{b}\| \leq K' + (|e| + K) \cdot B' \leq K + [2^{\gamma r} + r] \cdot B' =: B''$ this bound we call B'' . If the entries of \mathbf{b} are $(b_1, \dots, b_r, u_0, \dots, u_{N-1})$ then let the polynomial $H := \sum u_j \cdot x^j$. Now $0 < \text{Res}(H, f) \leq (B'' \cdot \|f\|_2)^N$ and $p^A | \text{Res}(H, f)$. Thus $p^A \leq (B'' \cdot \|f\|_2)^N = 2^{\mathcal{O}(N(N+H))}$. So set the upper limit to $\log_p(B'' \cdot \|f\|_2)^N$, and we must have a successful call to Algorithm 3 or have solved the problem. This means that if $p^A > (B'' \cdot \|f\|_2)^N = 2^{\mathcal{O}(N(N+H))}$ and the bound on the number of LLL calls is reached the algorithm must have terminated and by the previous section the output must be correct.

Overall Complexity version 1

The rest:

We have to compute only N CLD bounds (which are all done with low precision floats anyhow).

Then each Hensel lift triggers $\mathcal{O}(1)$: 1) matrix multiplications for checking 2) CLD computations using power series techniques

Each LLL call triggers, up to $\mathcal{O}(1)$ 1) and 2) again and a single call to check if solved [later we'll toss in a Gram matrix update too]

In the final loop we have r CLD computations using all N coefficients (not power series techniques) and N matrix multiplications all at full precision $N(N + H)$ -bits. (In the previous loops only $\mathcal{O}(1)$ calls (up to the LLL decisions... FIXME: commit to using more CLD's or not, in face this whole thing might cost more if we tackle huge examples, so far $\mathcal{O}(1)$ has been enough but maybe in the big guys those heuristics will change) of these things are used.)

First let's tackle the LLL worst-case costs: There are $\mathcal{O}(r^2)$ calls to LLL, with (thanks to the virtualization technique) $\mathcal{O}(r)$ entries per row to update, and each entry has bit-size of $\mathcal{O}(r)$.

We want to analyze the cost of a single call to fpLLL which uses τ switches. Such an analysis is given in Theorem 2 of JOURNALFPLL: $\mathcal{O}(n(\tau + d \log(dB))(d + \log(B))\mathcal{M}(d))$. We did a slight modification for our case in which we show that we can replace $d \log(dB)$ by d . We reason by the following, Theorem 5 in the same paper gives the cost of the t^{th} iteration as $\mathcal{O}(dn(d + \log B)(d + \log M(t)))$ and lemma 7 gives $\log M(t) \leq \mathcal{O}(d) + \log(\|b_{\kappa(t)}^{(t)}\|) - \log(\|b_{\alpha(t)}^{(t)}\|)$. The remainder of the analysis in the fpLLL journal paper goes into reducing the impact of this

term by canceling and so on (in order to reduce the dependency on $\log(B)$). In our situation we already know that the size of the vectors throughout the call to LLL is $\mathcal{O}(d)$ bits, as the vectors below the current index are LLL-reduced already and the inputs had $\mathcal{O}(r)$ bits, (simply use Theorem 5 of fpLLL journal paper again if you'd like). Thus the cost per iteration in this situation is $\mathcal{O}(d^4)$ and the number of iterations is $\mathcal{O}(d+\tau)$. Note that we needed the virtualization of the entries to reduce n to $\mathcal{O}(r)$.

Now if we say that the j^{th} call to LLL uses τ_j switches then the total cost of LLL throughout the algorithm is $\sum_{j=1}^{\mathcal{O}(r^2)} (r^3 \mathcal{M}(r) + \tau_j(r^2) \mathcal{M}(r))$ and we've shown that $\sum_j \tau_j = \mathcal{O}(r^3)$ so the total cost is $\mathcal{O}(r^7)$ or $\mathcal{O}(r^5 \mathcal{M}(r))$.

Next we can look at the cost of Hensel lifting, which is unaffected by our stopping and starting thanks to the storing of the inverse trees. The final precision is $\mathcal{O}(N(N+H))$, so we refer to MCA to get a total Hensel cost of $\mathcal{O}(\mathcal{M}(N) \log r \mathcal{M}(N[N+H] \log p))$ in terms of finding a suitable prime we can refer to the original LLL paper which reasons that a prime of size $\log p = \mathcal{O}(\log N + \log(H + \log N))$ must exist. This is the term in our complexity which is most impacted by the use of 'fast' multiplication (which we have in FLINT/MPIR) namely in classical arithmetic this cost is $\mathcal{O}(N^4(N+H)^2 \log r \log^2 p)$ where as with fast arithmetic this is $\mathcal{O}(N^2(N+H) \log r \log p \log(N^2(N+H) \log r \log p))$. These costs dominate the cost of factoring modulo p .

There are at most N computations of the CLD bound which could be done with a floating point poly root finder on degree N (the coeffs could be floats too). This can't be much but I haven't written it down.

In each non-final loop of algorithm 2 we compute $\mathcal{O}(r)$ CLD divisions which are done with power series modulo p^a of degree $\mathcal{O}(1)$. There are $\mathcal{O}(\log(N(N+H)))$ loops. Suppose k is the number of CLDs being computed in a loop of Hensel lifting (so that I can put off this $\mathcal{O}(1)$ thing until some more experimenting to determine the best way), then the sum costs of computing these CLDs is $\sum_{i=1}^{\log(N(N+H))} r(k \cdot (H + 2^i)^2) = \mathcal{O}(\log(N(N+H)) H^2 r k + (N(N+H))^2 r k)$. (This is a classical analysis.)

In the worst-case final loop we need all N CLD computations of the r factors and each local factor has are $\mathcal{O}(N(N+H))$ bits and full degree's are needed, that is r computations of $f \cdot f'_i / f_i$ modulo p^a .

Here we'll need to amortize the analysis a bit so let's say that d_j is the degree of f_j then $\sum d_j = N$, f has H -bit coeffs. First we compute $f \cdot f'_j$ which involves $(N+1) \cdot d_j$ multiplications with one operand having H bits the other having $N(N+H)$ bits for a cost of $\mathcal{O}(N \cdot d_j \cdot NH(N+H))$ (classical multiplication here). We can then reduce these coefficients modulo p^a and this product has degree $N + d_j$. Now we divide this product by f_j modulo p^a , now we know that this division can be done exactly and that the f_j is effectively monic, so we do $N + d_j$ multiplications of f_j by modular inverses of the leading coefficient of what is left followed by subtraction. These operations cost $\mathcal{O}((N + d_j) \cdot d_j \cdot (N(N+H))^2)$ which dominates the cost of the earlier multiplication. So if we sum over the d_j we get $\sum_{j=1}^r (N + d_j) d_j (N(N+H))^2 = \mathcal{O}(N^4(N+H)^2)$.

We also do $\mathcal{O}(1)$ matrix multiplications of U by these CLD vectors, except in the final loop we do $\mathcal{O}(N)$ such multiplications. Each such matrix multiplication involves $\mathcal{O}(r^2)$ multiplications and additions of numbers with $\mathcal{O}(r)$ -bits and others with $\log p^a$ bits. So $\mathcal{O}(r^3 N^2(N+H))$ in the final loop which dominates the rest of the loops.

Throughout the algorithm there are $\mathcal{O}(r^2)$ updates of the Gram matrix each involving $\mathcal{O}(r^2)$ multiplications and additions of $\mathcal{O}(r)$ -bit numbers, which is absorbed by the LLL costs.

So the final loop dominates the costs of the non-LLL parts (and this final loop has never been reached in practice) giving the worst-case complexity using classical multiplication of $\tilde{\mathcal{O}}(r^7 + N^4(N + H)^2)$ (we've dropped small log factors in the Hensel lifting estimates).

To adjust to 'fast multiplication' would drop the LLL costs to $\tilde{\mathcal{O}}(r^6)$ Hensel lifting to $\tilde{\mathcal{O}}(N^2(N + H))$, the matrix multiplications drop only a factor r to $\tilde{\mathcal{O}}(r^2N^2(N + H))$, leaving the CLD computations which are $\tilde{\mathcal{O}}(N^3(N + H) + N(N + H)r \cdot k)$ (the N^2 multiplications in that modular division might be able to drop to $N \log(N)$ making this $N^2(N + H)$ plus the rest). So (barring $k = r^2$) we have the fast multiplication costs at $\tilde{\mathcal{O}}(r^6 + [r^2 + N]N^2(N + H))$.

In terms of expected cost we think that at most $N + H$ bits will be needed before the problem is solved and typically only $\mathcal{O}(r)$ or $\mathcal{O}(1)$ calls to LLL or any of the smaller programs. This would lead to costs of something like $\tilde{\mathcal{O}}((N + H)(N + r^2) + r^6)$ with fast arithmetic (which FLINT has). Further these lattices are knapsack types which means that the LLL costs could follow the observations of Stehlé in LLL25 which hint that knapsack lattices could see behavior more like $\tilde{\mathcal{O}}(r^5)$ for smaller values of r .

FIXME: add a table showing the important inequalities (and their reason) and variable-ize the precision. For checking the inequalities it would be helpful to have a spread sheet that automates some of the checking.