Министерство науки и высшего образования Российской Федерации ФГАОУ ВО «Балтийский федеральный университет имени Иммануила Канта»

УТВЕРЖДАЮ
ректор БФУ им. И. Канта
_____ А.А. Фёдоров
«__»_____ 2022 года

Отчет
о научно-исследовательской работе _____ этап
“Алгоритмы нахождения короткого вектора в алгебраических решетках”
(договор на выполнение НИР № 2165 от 8 октября 2021 года)

Исполнитель _____

Согласовано:
Проректор по научной работе
_____ М.В. Дёмин

Руководитель проекта
_____ Е.А. Киршанова

Калининград
2022

Просеивание с использованием быстрого преобразования Фурье

А.С. Каренин

29 апреля 2022 г.

1 Введение

Задача поиска коротких ненулевых векторов решётки, а именно нахождения вектора длины, не превышающей более чем в $\gamma \in (1, \infty)$ раз длину кратчайшего ненулевого вектора в данной решётке (далее - approx-SVP), считается в общем случае сложной для фактора $\gamma = \text{poly } n$, где n - размерность решётки [May99]. Благодаря этому в 90-х годах прошлого века Jeffrey Hoffstein, Jill Pipher и Joseph H. Silverman предложили криптосистему NTRU [HPS98]. Её защищённость основывается на предположении о сложности approx-SVP для полиномиального фактора аппроксимации $\gamma = \text{poly } n$.

Для решения approx-SVP с фактором γ в решётке полной размерности можно вызвать блочный алгоритм редукции решёток, работа которого основана в том числе на нахождении кратчайшего вектора в подрешётке. Такой алгоритм, возвращающий короткие векторы в подрешётке называется SVP-оракул. Если он должен возвратить ненулевой вектор, не превышающий по длине кратчайший ненулевой не более, чем в γ раз, то этот алгоритм называется approx-SVP-оракул.

Для сокращения объёма информации, необходимой для хранения ключей, последние[1] представляют из себя многочлены из кольца $\mathbb{Z}[x]/P(x)$, где $P(x)$ - многочлен степени d. Обычно в качестве многочлена $P(x)$ выбирают некоторый циклотомический многочлен, задающий циклотомическое после простой степени, или многочлен $x^d - 1$, где d является степенью числа 2 [DWZ18]. Решётки, соответствующие таким инстанциям, называются алгебраическими. Более формальное и подробное объяснение понятия алгебраических решёток будет дано в разделе 2.1.

Естественным образом возникает вопрос: может ли описанное в прошлом абзаце свойство быть использовано не только для сокращения длин ключей, но и для атаки на криптосистемы, связанные с алгебраическими решётками?

В данной работе рассматривается подход к получению большого количества сравнительно коротких векторов алгебраической решётки, использующий алгебраическое свойство таких решёток, а именно тот факт, что ротации (умножение на элемент x кольца $\mathbb{Z}[x]/P(x)$) векторов также лежат в решётке. Иными словами, в данной работе предлагается концепция алгоритма реализации approx-SVP оракула с настраиваемым фактором аппроксимации, использующего вышеупомянутое свойство алгебраических решёток.

[1] кто здесь последние?

2 Предварительные сведения

2.1 Решётки

Решёткой называется дискретная аддитивная подгруппа $\mathbb{R}^n$ или $\mathbb{C}^n$ для некоторого натурального n.

Проекцией множества A на множество B называется отображение $\pi : A \cup B \to B$, обладающее свойством идемпотентности, а именно $\forall i \in \mathbb{N} : \pi = \pi^i$, и такое, что его образом является подмножество в B. Решётка Λ называется подрешёткой (относительно решётки $\mathcal{L}$), если $\Lambda \cap \mathcal{L} = \Lambda$.

Базисом $\boldsymbol{B}$ решётки $\mathcal{L}$ называется матрица, строки которой состоят из координат базисных векторов этой решётки. Определителем решётки называется значение $\det(\mathcal{L}) = \sqrt{\det \boldsymbol{B} \cdot \boldsymbol{B}^t}$.

Пусть $\mathcal{O}_F$ является кольцом целых циклотомического поля F с примитивным корнем из 1 степени f равным ζ_f. Тогда мы полагаем $x = \zeta_f$ и вводим понятие Эрмитово сопряжения: мы говорим, что Эрмитово сопряжение $*$ в этом кольце оставляет Z-коэффициенты неизменными и лишь заменяет x^i на x^{d-i}. В таком случае степень этого кольца (количество элементов в базисе F над $\mathbb{Q}$) есть $d = \varphi(f)$, где φ - функция Эйлера. Иными словами: $\left(\sum_{0 \leqslant i < d} c_i \cdot \zeta_f^i\right)^* = \sum_{0 \leqslant i < d} c_i \cdot \zeta_f^{f-i}$.

Матрицы могут быть заданы над кольцом $K = \frac{\mathbb{Q}[x]}{x^d - 1}$, обладающим системой образующих $\{x^i\}_{0 \leqslant i < d}$ и в этом случае мы просто полагаем $\left(\sum_{0 \leqslant i < d} c_i \cdot x^i\right)^* = \sum_{0 \leqslant i < d} c_i \cdot x^{d-i}$. Тогда степень d такого кольца (совпадающая со степенью его подкольца $\mathcal{O}_K = \frac{\mathbb{Z}[x]}{x^d - 1}$) равна количеству элементов в базисе над $\mathbb{Q}$ (над $\mathbb{Z}$ в случае подкольца $\mathcal{O}_K$). Такое кольцо K называется кольцом конволюций.

Для корректности дальнейших определений нам нужны квадратные положительно определённые матрицы. Для этого строится матрица $\boldsymbol{G} = \boldsymbol{B} \cdot \boldsymbol{B}^*$, где оператор * означает транспонирование и последующее Эрмитово сопряжение.

Пусть $\boldsymbol{B}^t = \boldsymbol{Q} \cdot \boldsymbol{R}$ - QR-декомпозиция матрицы $\boldsymbol{B}^t$ размера $n \times n$. Назовём GSO декомпозицией матрицы $\boldsymbol{B}$ следующее разложение:

$$\boldsymbol{B} = \boldsymbol{L} \cdot \tilde{\boldsymbol{B}}, \tag{1}$$

где $\boldsymbol{L} = \boldsymbol{R}^t \cdot \mathrm{diag}(\boldsymbol{R_{i,i}})_{0 \leqslant i < n}^{-1}$ - нижнетреугольная матрица с единицами на диагонали, а $\tilde{\boldsymbol{B}} = \mathrm{diag}(\boldsymbol{R_{i,i}})_{0 \leqslant i < n} \cdot \boldsymbol{Q}^t$.

Пусть F - циклотомическое поле степени d элементы которого задаются многочленами степени $d - 1$ от формальной переменной x. Пусть $\mathcal{O}_F$ - кольцо целых поля F. Такой $x \in \mathcal{O}_F$будем называть *основанием системы образующих кольца* $\mathcal{O}$ так как каждый элемент кольца выражается через линейную комбинацию над $\mathbb{Z}$ элементов множества $\{x^i\}_{0 \leqslant i < d}$. Назовём мощность d такой системы образующих индексом или степенью кольца $\mathcal{O}_F$.

В условиях уравнения 1 назовём LDL^* декомпозицию положительно определённой матрицы $\boldsymbol{G} = \boldsymbol{B} \cdot \boldsymbol{B}^*$ разложение следующего вида:

$$\boldsymbol{G} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^*, \tag{2}$$

где $\boldsymbol{D}$ - диагональная матрица, а $\boldsymbol{L}$ - нижнетреугольная с единицами на диагонали. Несложно убедиться, что $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{R}_{i,i}^2)_{0 \leqslant i < n}$.

Преобразованием Фурье элементов из кольца конволюции K степени d называется отображение $\mathcal{F} : K \to \mathbb{C}^d : \sum_{i=0}^{d-1} x^i \cdot c_i \mapsto (\sum_{i=0}^{d-1} \zeta_d^i \cdot c_i)$, где ζ_d - d-й примитивный корень из 1. Это отображение, делёное на $\sqrt{d}$ является изометрией, так как DFT-матрица, делёная на $\sqrt{d}$ унитарна - это можно можно проверить, перемножив её с собой же, но транспонированной и сопряжённой и получить единичную).

Так как на основном этапе алгоритма мы будем работать не с циклотомическим полем F, имеющим кольцо целых $\mathcal{O}_F$ и расширение скаляров $F \otimes \mathbb{R}$, а с кольцом конволюций K степени d, то введём аналоги кольца целых и расширения скаляров для таких колец как $\mathcal{O}_K = \mathbb{Z}[x]/(x^d - 1)$ и $K \otimes \mathbb{R}$ соответственно.

Алгебраической решёткой называется свободный модуль $\mathcal{L}$ ранга k над некоторым кольцом $\mathcal{O}$, подразумевающим вложение, аналогичное вложению Минковского в $\mathbb{C}^{kd}$ для некоторого натурального d. В качестве таких колец мы будем использовать $\mathcal{O} = \frac{\mathbb{Z}[x]}{P(x)}$, где $P(x)$ - либо циклотомический многочлен, либо многочлен вида $x^n - 1$, где $n \in \mathbb{N}$.

Стоит отметить, что, так как вложение Минковсого является подстановкой корней из 1 в формальную сумму степеней x, оно, по сути дела, является преобразованием Фурье.

Для вектора $v = (v_0, \ldots, v_{k-1})$ над кольцом $\mathcal{O}$ введём понятие ротации. i-й ротацией вектора v называется вектор $x^i v = (x^i v_0, \ldots, x^i v_{k-1})$. В алгебраических решётках ротации векторов всегда принадлежат исходным решёткам.

2.2 Log-unit решётки и лемма о связи алгебраических и геометрических норм

Пусть F - числовое поле, Q - его подполе, $G = \mathrm{Gal}(F/Q)$ - их относительная группа Галуа. Нормой $N_{F/Q}$ элемента $\kappa \in F$ называется произведение $\prod_{\sigma(k)\in G} \sigma(\kappa)$. Аналогично, назовём следом $N_{F/Q}(\kappa)$сумму $\sum_{\sigma\in G} \sigma(\kappa)$.

Для элемента κ из числового поля F степени d определим отображение $\mathrm{Log}(\kappa)$ как $(\log|\sigma_0(\kappa)|, \log|\sigma_1(\kappa)|, \ldots, \log|\sigma_{d-1}(\kappa))|$, где $\sigma_i \in \mathrm{Gal}(F/\mathbb{Q})$. Для всех элементов $u \in K$ алгебраической нормы 1 (такие u называются единицами) построим множество $\Lambda = \mathrm{Log}\{u \,|\, N_{F/\mathbb{Q}}(u) = 1\}$. Это множество является дискретной аддитивной подгруппой в $\mathbb{C}^d$, то есть, решёткой. Обозначим за $H = \{x \in \mathbb{R}^d \,|\, \sum_{i=0}^{d-1} x_i = 0\}$ гиперплоскость. Нетрудно показать, что $\Lambda \subset H$, а сама гиперплоскость ортогональна вектору $\mathbf{1} = (1, 1, \ldots, 1)$.

Далее вместо $N_{F/\mathbb{Q}}$ будем писать N, когда понятно о какой алгебраической норме идёт речь.

Лемма 2.1 ([PMHS19], лемма 2.11). *Пусть $\kappa \in F$ - элемент числового поля F. Пусть также h - ортогональная проекция $\mathrm{Log}(\kappa)$ на Log-unit-решётку. Тогда справедливо:*

$$\mathrm{Log}(\kappa) = h + \frac{\log|N(k)|}{d}\mathbf{1} \tag{3}$$

Тогда по предыдущей лемме для любого $\kappa \in F$ верно: $\mathrm{Log}(\kappa) = h + \frac{\log|N(k)|}{d}\mathbf{1}$, где h является ортогональной проекцией $\mathrm{Log}(\kappa)$ на гиперплоскость H.

Лемма 2.2 ([PMHS19], лемма 2.12). *Для любого элемента κ из числового поля F верно:*

$$||\kappa||_2 \leqslant \sqrt{d}\,|N(\kappa)|^{1/d}\, 2^{||h||_2} \tag{4}$$

Лемма 2.3 ([KEF19]). *Пусть F - циклотомическое поле степени $d = \varphi(f)$ с кондуктором f и кольцом целых $\mathcal{O}_F$, $\kappa \in F \otimes \mathbb{R}$. Тогда существует квазилинейный (в данном случае сложности не более, чем $O(d\log(r))$, где r - наибольший простой делитель d) алгоритм нахождения такой единицы $u \in \mathcal{O}_F^\times$, что для любого вложения числового поля $\sigma : F \to \mathbb{C}$ верно:*

$$||\sigma(\kappa \cdot u^{-1})|| \leqslant 2^{O(\sqrt{f \log f})}\, N(\kappa)^{1/d} \tag{5}$$

2.3 KEF-LLL алгоритм

Пусть $\boldsymbol{B} \in \mathcal{O}_F^{n\times n}$ - матрица, задающая базис алгебраической решётки ранга n над циклотомическим полем F степени d. Назовём псевдо LLL-редуцированным базис $\boldsymbol{B'}$, если он получен на выходе алгоритма KEF-LLL: $\boldsymbol{B'} = \text{KEF-LLL}(\boldsymbol{B})$ [KEF19].

Лемма 2.4 ([KEF19], Грам-Шмидт профиль базиса). *Пусть $\boldsymbol{B} = \boldsymbol{L}\cdot\tilde{\boldsymbol{B}}$ - псевдо-LLL редуцирован. Тогда $\forall 0 \leqslant i < n$ справедливо:*

$$N(\tilde{\boldsymbol{B}}_j) \leqslant 2^{d^2 \log^{O(1)} d}\, \sqrt{N(\tilde{\boldsymbol{B}}_j) \cdot N(\tilde{\boldsymbol{B}}_{j+1}))} \tag{6}$$

2.4 Операторы матрификации и векторизации, вложение Дюки-Преста

Алгоритм, предложенный в данной статье работает над кольцом конволюций, однако, на вход ему подаются решётки над циклотомическими полями. В этом подразделе мы объясняем, как отображаться между этими кольцами, а также вводим другие важные для описания алгоритма понятия: операторы матрификации и векторизации.

Пусть $\mathcal{F}$ - циклотомическое поле кондуктора f и степени $d' = \varphi(f)$. Тогда [DP16] существует такое кольцо конволюции K степени $d, d'|d$, что $\mathcal{F}$ можно инъективно вложить в K. Рассмотрим построение этого вложения.

Введём вспомогательные многочлены:

$$\phi(x) = \prod_{\zeta\in\Omega_{d'}} (x-\zeta) = \prod_{k\in\mathbb{Z}_{d'}^\times} (x-\zeta_{d'}^k) \tag{7}$$

$$\psi(x) = \prod_{\zeta\notin\Omega_d, \zeta^{d'}=1} (x-\zeta) = \prod_{k\in\mathbb{Z}_d\setminus\mathbb{Z}_d^\times} (x-\zeta_{d'}^k), \tag{8}$$

где $\Omega_{d'}$ - множество примитивных корней степени d' из единицы.

Смысл этих вспомогательных многочленов в том, что $\phi(x)$ задаёт циклотомическое поле, а $\phi(x)\cdot\psi(x)$ - задаёт кольцо конволюций, в которое мы будем вкладывать элементы из первого.

Тогда существует элемент e_d такой, что $e_d \equiv 1 \pmod{\phi(x)}$ и $e_d \equiv 0 \pmod{\psi(x)}$. Назовём вложением Дюки-Преста следующее отображение:

$$\mathcal{I} : \mathcal{K} \to K \ ; \ f \mapsto f \cdot e_d. \tag{9}$$

Вложение Дюки-Преста является инъективным гомоморфизмом колец и изометрией[DP16], а, значит, и биекция $\mathcal{K}$ на его образ. Например, вектор $(1, 2, 3, 4)$, соответствующий элементу $1 + 2x + 3x^2 + 4x^3$ кольца целых $\mathcal{O}$ циклотомического поля $F = \mathbb{Q}[x]/(x^4 + 1)$, можно вложить в кольцо конволюций $K = \mathbb{Q}[x]/(x^8 - 1)$ и образом этого вложения будет вектор $(1/2, 1, 3/2, 2, -1/2, -1, -3/2, -2)$.

Пусть $a \in K^d$ - вектор длины d над кольцом конволюций, а натуральное d' делит d. Пусть $k = d/d'$ Обозначим за x формальную переменную кольца K как свободного $\mathbb{R}$-модуля ранга d, а $y = x^k$. Положим $\mathrm{gpd}(d)$ - наибольший собственный делитель числа d. Определим оператор векторизации $\mathcal{V}_{d/d'} : K_d^m \to K_{d'}^{km}$ рекурсивно:

- Для $d = d' = 1$ получаем тождественное отображение.
- Для $d = \mathrm{gpd}(d)$ и элемента $a \in K$ можно перестановкой слагаемых записать $\sum_{i \in \mathbb{Z}_k} x^i a_i(y)$, где $a_i \in K_{d'}$ для всех i. Тогда

$$\mathcal{V}_{d/d'}(a) := (a_0, \ldots, a_{k-1}). \tag{10}$$

- Для вектора $v \in K_d^m$ оператор применяется покомпонентно.
- Для $d''|d'|d$ и вектора $v \in K_d^m$:

$$\mathcal{V}_{d/d''}(v) = \mathcal{V}_{d'/d''} \circ \mathcal{V}_{d/d'}(v), \tag{11}$$

  где $\circ$ означает композицию отображений.

Аналогично подойдем к определению оператора матрификации $\mathcal{M} : K_d^{m \times n} \to K_d'^{km \times kn}$:

- Для $d = d' = 1$ получаем тождественное отображение.
- Для $d = \mathrm{gpd}(d)$ и элемента $a \in K$ результатом применения оператора матрификации к элементу a является:

$$\begin{pmatrix} \mathcal{V}_{d/d'}(a) \\ \mathcal{V}_{d/d'}(xa) \\ \vdots \\ \mathcal{V}_{d/d'}(x^{k-1}a) \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & \ldots & a_{k-1} \\ ya_{k-1} & a_0 & \ldots & a_{k-2} \\ \ddots & \ddots & \ddots & \ddots \\ ya_1 & ya_2 & \ldots & a_0 \end{pmatrix} \tag{12}$$

Операторы векторизации и матрификации являются инъективными линейными морфизмами алгебр. Также, они являются изометриями. Несмотря на сложное определение, всё, что они делают - перестанавливают координаты векторов. Важным также является факт, доказанный в статье [DP16] о том, что, если $d'|d$ и $\kappa = d/d'$, то верно:

$$a \in \mathcal{I}(F) \iff \mathcal{V}_{d/d'}(a) \in \mathcal{V}_{d/d'}(\mathcal{I}(F))^\kappa, \tag{13}$$

что говорит о том, что операторы $\mathcal{I}$ и $\mathcal{V}_{d/d'}$ коммутируют.

Также в статье Дюки и Преста[DP16] предлагаются два алгоритма: составления LDL^* дерева и ffNearestPlane алгоритма, использующего ускорение при помощи быстрого преобразования Фурье. Рассмотрим их суть.

LDL^* деревом матрицы $\boldsymbol{G} = \boldsymbol{B}\boldsymbol{B}^*$ называется дерево в узлах которого на i уровне находятся матрицы $L_{i,j}$, помогающие ускорить ffNearestPlane. В его листьях хранятся квадраты длин векторов Грама-Шмидта, расположенные в reverse digit order порядке (то есть, определённым образом переставленные).Смысл этого дерева - ускорение умножения на матрицу L для перехода к базису, образованному векторами Грама-Шмидта.

Пусть d - степень кольца конволюций K.Суть алгоритма построения LDL^* дерева состоит в рекурсивном LDL^* разложении матриц. В корне дерева производится LDL^* декомпозиция исходной матрицы $\boldsymbol{B} \in \mathcal{O}_K^{k \times k}$ над кольцом $K_{\mathbb{R}} = K \otimes \mathbb{R} = \frac{\mathbb{Q}[x]}{x^d - 1} \otimes \mathbb{R}$. Далее имеем $d/d' = d/\mathrm{gpd}(d)$ потомков - каждый соответствует элементу $\boldsymbol{D}_{i,i} \in \mathcal{O}_K, 0 \leqslant i < d$

диагональной матрицы $\boldsymbol{D}$ в LDL^* разложении. Для этих потомков строится матрификация $\mathcal{M}_{d/d'}$ и рассматривается как матрица $\boldsymbol{B'}$, после чего происходит очередная LDL^* декомпозиция, но уже матрицы $\boldsymbol{B'}$ над подкольцом конволюций. Процесс продолжается рекурсивно, пока не будет достигнуто кольцо $\mathbb{R}$.

Суть алгоритма ffNearestPlane заключается в том, что при умножении на матрицу $\boldsymbol{L}$ различных элементов колец конволюции в процессе работы оригинального (не использующего быстрое преобразование Фурье) Nearest Plane алгоритма часто происходят "лишние" умножения: пусть стоит цель умножить векторы (u, v) и (u, w), представляющие собой векторизацию каких-то двух исходных векторов над кольцом конволюций, на матрицу $\boldsymbol{L} = \begin{pmatrix} 1 & 0 \\ L_{1,0} & 1 \end{pmatrix}$. В таком случае нам не нужно умножать одинаковую для обоих векторов компоненту u на соответствующие компоненты из $\boldsymbol{L}$ - достаточно сделать это один раз и запомнить. Примерно то же самое и делает LDL^* дерево, как бы запоминая это умножение.

3 Основная идея алгоритма

Основной идеей алгоритма является стратегия "разделяй и властвуй" , а также активное использования факта принадлежности ротаций векторов решётке и быстрого умножения при помощи преобразования Фурье. Немалую роль в алгоритме является алгоритм NearestPlanes - обобщение алгоритма Babai Nearest Plane.

Пусть на вход подаётся решётка $\mathcal{L}$ ранга k над циклотомическим полем $\mathbb{K}_d$. Алгоритм разделяется на 4 основных этапа:

3.1 Предобработка

Пусть натуральное δ означает во сколько раз степень кольца конволюции, в которое можно вложить циклотомическое поле при помощи вложения Дюки-Преста, больше степени циклотомического Для циклотомических полей степени двойки $\delta = 2$.

Векторы Грама-Шмидта тем качественней, чем ближе они друг к другу по длине. Для улучшения качества векторов Грама-Шмидта на базисе алгебраической решётки ранга k над циклотомическим полем F степени d/δ вызывается алгебраический LLL алгоритм, описанный в [KEF19]. Чем больше параметр δ, тем больший ранг алгебраической решётки, но тем меньше степень циклотомического поля, над которым работает алгоритм.

Для улучшения качества выхода алгоритма KEF-LLL алгоритм можно вызывать не на алгебраической решётке ранга k над полем степени d, а на соответствующей алгебраической решётке ранга κk над полем степени $d/(\kappa\delta)$ для некоторого κ, делящего d/δ.

Полученный базис решётки при помощи отображения матрификации Дюки-Преста и одноимённого вложения вкладывается в кольцо матриц размера $k \times k$ над кольцом конволюций K степени d. После этого для полученной матрицы $\mathbf{B}$ высчитывается LDL^* дерево, которое впоследствии будет использоваться для подъёма векторов из проективных подрешёток.

3.2 Генерация проективных подрешёток

Для выбранной длины блока β генерируются $k\left(\frac{d}{\beta}\right)^2$ проективных подрешёток, имеющих тривиальное пересечение (а, именно, вектор $\mathbf{0}$). Это возможно так как алгебраическую решётку $\mathcal{L} = \sum_{i=0}^{k-1} v_i \cdot \mathcal{O}_{K_d}$ ранга k над K_d можно представить как алгебраическую решётку ранга $\left(\frac{kd}{\beta}\right)$ над K_β следующим образом: $\mathcal{L} = \sum_{i=0}^{k-1}\sum_{j=0}^{\frac{d}{\beta}-1} x^j \cdot v'_i \cdot \mathcal{O}_{K_\beta}$, где $v_i \in K_d^k$ - базисные вектора исходной решётки и $v'_i \in K_\beta^{d/\beta}$ - базисные вектора той же решётки, но над подполем, а x - основание системы образующих кольца $\mathcal{O}_{K_\beta}$. Каждое слагаемое будет представлять собой одну из вышеупомянутых проективных подрешёток (ранга 1 над $\mathcal{O}_{K_\beta}$, но большего ранга над подкольцом).

Пусть $\boldsymbol{B} \in \mathcal{O}_K^{k\times k}$ - матрица, представляющая базис решётки ранга k над кольцом $\mathcal{O}_{K_d}$. Применяя оператор матрификации $\mathcal{M}_{d/(\beta)}(\boldsymbol{B}) = \boldsymbol{B'}$ мы получаем матрицу решётки ранга $\frac{kd}{\beta}$ как в предыдущем абзаце. Её элементы (после составления матрицы ротаций) сами создают базисы $\frac{kd}{\beta}$ решёток ранга $\beta/\mathrm{gpd}(\beta)$ над подкольцами кольца $\mathcal{O}_{K_\beta}$, естественным образом группируя элементы описанным в прошлом абзацем образом.

3.3 many-approx-SVP's в проективных подрешётках

Каждой из вышеупомянутых проективных решёток размерности β над $\mathbb{Z}$ ставится в соответствие список из $2^{\Theta(\beta)}$ её коротких векторов. Желательно, чтобы среди них был кратчайший ненулевой. Для этого можно вызвать слегка модифицированный алгоритм перечисления, возвращающий не кратчайший вектор, а $2^{\Theta(\beta)}$ кратчайших векторов.

3.4 Подъём и дальнейшее просеивание найденных решений

Вышеупомянутые решётки должны быть выбраны так, чтобы они соответствовали одинаковому уровню LDL^* дерева. Первый уровень LDL^* дерева (корень будем считать нулевым) имеет k узлов. Соответственно, у нас есть k^2 проективных решёток ранга 1 над $\mathcal{O}_{K_d}$. Он соответствует состоянию списков перед последним слиянием.

Эти каждый из этих списков в свою очередь состоит из слияния $d/\mathrm{gpd}(d)$ списков векторов, соответствующих проективным подрешёткам ранга 1 над $\mathcal{O}_{K_{\mathrm{gpd}(d)}}$.

Как только все векторы из списков подняты при помощи алгоритма NearestPlanes, проводится слияние списков - для каждого из $\mathrm{ndm}(\beta, d)/\beta$ списков (исключение: на последнем шаге слияний списков ровно k) проводится процедура слияния (в результирующий список попадают наикратчайшие из сумм ротаций пар [2]), после чего решение добавляется в новосозданный список, поставленный в соответствие проективной подрешётке - предку. Поле этого наш размер блока β становится больше, а именно $\beta := \mathrm{ndm}(\beta, kd)$ (кроме последнего шага-исключения - на последнем шаге размерность становится размерностью оригинальной решётки) и процедура слияния повторяется для всех решёток - предков, пока β не станет равной kd. На выход подаются все векторы из единственного оставшегося списка, соответствующего исходной решётке.

Само слияние это не просто объединение списков векторов L_1 и L_2. Для каждой пары $(u, v) \in L_1 \times L_2$ в результирующий список добавляется кратчайший из них и

[2] Также можно решать задачу k-list-minimum, увеличив эффективность слияния с точки зрения длины векторов

кратчайшую из сумм $v + x^i u$, где $0 \leqslant i < d - 1$. Таким образом, векторы из результирующего списка не могут быть длиннее исходных. Напротив, мы ожидаем, что они будут короче.

Заметим, что подсчитать нормы $||v+x^i u||$ векторов длины β, где $0 \leqslant i < d-1$ можно тривиально вычислив соответствующие значения за $O(d^2)$ операций. Однако, векторы, имеющие большое внутреннее произведение, могут иметь малый угол между собой, а, следовательно, давать короткую сумму. Для всех $0 \leqslant i < d - 1$ можно посчитать d внутренних произведений вида $\langle u, x^i \cdot v \rangle$ можно за $O(d \log d)$ операций, если нам даны векторы u', v' - представления u, v в домене Фурье. А именно, можно умножить $u \circ \bar{v}$ и получить образ вектора внутренних произведений в домене Фурье. Далее можно за $d \log d$ вернуться в домен времени и оценить каждое внутреннее произведение.

4 Алгоритм

4.1 Тело алгоритма

Пусть на вход подан базис B свободного модуля ранга k над кольцом $\mathcal{O}_{K_d}$[3], задающий алгебраическую решётку. Ниже приведён псевдокод алгоритма, возвращающего короткие векторы.

Algorithm 1 FFT Sieve saturation

Input: A pseudo-LLL-reduced basis $\mathbf{B} \in \mathcal{O}_{K_d}^{k \times k}$ of a $\mathcal{O}_K$-lattice $\mathcal{L}$; $K_d = \frac{\mathbb{Q}[x]}{(x^d-1)}$, parameter β, s.t. $\beta \mid d$.
Output: An inner state (all variables and lists).

Preprocessing:

$\mathbf{B} := \mathcal{M}_{d/1} \left(\begin{pmatrix} \mathcal{F}(\mathbf{B}_{[0,0]}) & \mathcal{F}(\mathbf{B}_{[0,1]}) & \dots & \mathcal{F}(\mathbf{B}_{[0,k-1]}) \\ \mathcal{F}(\mathbf{B}_{[1,0]}) & \mathcal{F}(\mathbf{B}_{[1,1]}) & \dots & \mathcal{F}(\mathbf{B}_{[1,k-1]}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{F}(\mathbf{B}_{[k-1,0]}) & \mathcal{F}(\mathbf{B}_{[k-1,1]}) & \dots & \mathcal{F}(\mathbf{B}_{[k-1,k-1]}) \end{pmatrix} \right)$

$\mathbf{Tree} := \text{ffLDL}(\mathbf{B} \cdot \mathbf{B}^*)$

Generate $k \left(\frac{d}{\beta}\right)^2$ projected lattices:

for $0 \leqslant i, j < k \left(\frac{d}{\beta}\right)^2$ **do**
  for $0 \leqslant p, r < \beta$ **do**
    $\mathcal{L}_{[i,j]} := \mathbf{B}_{[i\beta \,:\, (i+1)\beta \;,\; j\beta \,:\, (j+1)\beta]}$
  end for
end for

many-approx-SVP's for sublattices:

for $0 \leqslant i, j < k \left(\frac{d}{\beta}\right)^2$ **do**
  $\mathrm{V}_{i,j} = \text{sample}(\mathbf{Tree}, \mathcal{L}_{[i,j]}, 2^{\Theta(\beta)})$ \\ vectors from $\mathcal{L}_{[i,j]}$ to sieve
  for $0 \leqslant l, r < |\mathrm{V}_{i,j}|, l < r$ **do**
    $u, v := \mathrm{V}_{i,j}[l], \mathrm{V}_{i,j}[r]$
    $\mathrm{V}_{i,j}[l], \mathrm{V}_{i,j}[r] := \text{short\_rot}(u, v)$
  end for
end for
return Algorithm's inner state

[3] $\mathcal{O}_{K_d}$ следует воспринимать как аналог кольца целых, но не путать с таковым, так как кольца конволюции - не являются полями.

После выполнения данного алгоритма мы прошли первые три шага, а именно: предобработку, генерацию проективных подрешёток и many-approx-SVP's в проективных подрешётках. Благодаря первому у нас есть псевдо-LLL редуцированный базис, необходимый для ограничения увеличения нормы векторов при подъёме при помощи алгоритма NearestPlanes. Второй шаг даёт нам набор проективных подрешёток, с каждой из которых можно параллельно работать, потом сливая их в меньшее количество более больших, что будет делаться в следующем алгоритме. И, наконец, 3 шаг сопоставляет каждой подрешётке набор коротких векторов.

Алгоритм 2 представляет собой последний шаг - слияние листов. Он получает на вход внутреннее состояние (все переменные и списки) Алгоритма 1. В обозначениях алгоритма соответствующие проективной подрешётке над $K_{d''}$ наборы векторов из d''/d' проективных подрешёток над $K_{d'}$ сливаются в k больших списков, ассоциирующихся с проективными подрешётками ранга 1 над $\mathcal{O}_{K_d}$. Далее, они поднимаются тем же алгоритмом ffNearestPlanes, но уже в исходную решётку. Затем происходит слияние. Так как мы заинтересованы в approx-SVP оракуле, мы возвращаем все векторы.

Отдельно стоит заметить, что, если найдется какой-либо вектор, не вписывающийся на данной итерации в границы по норме, его можно исключить из списка. Псевдокод этого алгоритма приведён ниже.

Algorithm 2 FFT Sieve

Input: inner state of Algorithm 1 at the end of running (all variables and lists)
Output: A short vector.

Merge all lists:

$d' = \beta$, $d'' = \mathrm{ndm}(d', d)$ \\$\mathrm{ndm}(a, c)$ is such least $b > a$ that $a \mid b$ and $b \mid c$
while $d'' < d$ **do**
  $\kappa := \left(\frac{d''}{d'}\right)^2$ \\$\kappa$ is number of lists merging into one
  for $0 \leqslant l, r < d/d''$ **do** \\ prepeare lattices for lifting
    $\left(\mathcal{L}'_{[l,r]}\right)_{[i,j]} := \left(\mathcal{L}_{[l\kappa+i, r\kappa+j]}\right)$ for all $0 \leqslant i, j < \kappa$
  end for
  $V' := \emptyset$
  for $0 \leqslant i, j < d/d'$ **do**
    $\mathrm{W}_{i,j} := \mathrm{lift}(\mathcal{L}'_{[i//\kappa, j//\kappa]}, \mathrm{V}_{i,j})$
    $\mathrm{V}'_{i//\kappa, j//\kappa} := \mathrm{sieve\_merge}(V'_{i//\kappa, j//\kappa}, W_{i,j})$
  end for
  $d' := d''$, $d'' := \mathrm{ndm}(d', d)$
  $\mathcal{L}, V := \mathcal{L}', V'$
end while
$V' = \emptyset$
for $0 \leqslant i, j < k$ **do**
  $\mathrm{W}_{i,j} := \mathrm{lift}(\mathcal{L}'_{[i//\kappa, j//\kappa]}, \mathrm{V}_{i,j})$
  $\mathrm{V}' := \mathrm{sieve\_merge}(V', W_{i,j})$
end for
return shortest vector from V'

4.2 Вспомогательные алгоритмы

Пусть нам даны два вектора u и v из кольца конволюций K_d с образующим основанием базиса x. Мы знаем, что $x^i \cdot v$ тоже принадлежит алгебраической решётке. Поэтому вместо нахождения $\min(\{u \pm v\})$ будем находить $\min(\{u + x^i \cdot v\}_{0 \leqslant i < d})$. Псевдокод этого алгоритма приведён ниже.

Algorithm 3 short_rot

Input: $u, v \in \mathcal{L} \subset \mathcal{O}^{k\times k}_{K'\subseteq K}$
Output: 2 short vectors.

1: **for** $0 \leqslant i < k$ **:** $v_i := \text{conjugate}(v_i) = \text{concatinate}(v_i[0], \text{reverse}(v_i[1:]))$
2: $L = u \circ v$ \\$\circ$ is a coordinate-wise product
3: $L := \mathcal{F}^{-1}(L)$
4: find such index l that L_l is maximized
5: $u, v := \min_{||\cdot||}(u, v), \max_{||\cdot||}(u, v)$
6: **return** $u, \min_{||\cdot||}(v, u + x^l \cdot v)$ \\$x$ is a formal variable of v as a polynomial

Заметим, что векторы u, v, подаваемые на вход данному алгоритму, представлены в домене частоты (это их образ при быстром преобразовании Фурье). Именно этот факт позволяет найти кратчайшую из сумм u с ротациями v за $O(kd)$.

Следующим алгоритмом является алгоритм слияния списков векторов. Он рандомизованно выбирает вектор w из списка W и заменяет его вместе с вектором $v \in V$ на пару коротких.

Algorithm 4 sieve_merge

Input: V, W - a lists of vectors of $\mathcal{L}$
Output: list V' of short vectors s.t. $|V'| = |V| + |W|$.

1: $V' := \emptyset$
2: **for** $0 \leqslant i < |V|$ **do**
3: $\quad V' := V' \cup \text{short\_rot}(V_i, W_t)$ for random $t \in [0, |V|]$
4: **end for**
5: $V' := V' \cup ((V \cup W) \setminus V')$
6: **return** V'

Для подъёма используется функция lift. Её основное предназначение - дематрификация. Результат подъёма возвращает функция NearestPlanes. Пусть $\mathbb{I}_{i<j}$ возвращает 1, если $i < j$ и 0 иначе.

Algorithm 5 lift

Input: vector $v \in \mathcal{L}_{i,j}$ of dimension d' over $\mathbb{Q}$, $\mathcal{L}'_{i//\kappa,j//\kappa}$ of dimension $d'' = d'\kappa$ over $\mathbb{Q}$, $C \in (\mathbb{Z}^+)^{d'}$.
Output: vector $u \in \mathcal{L}'_{i//\kappa,j//\kappa}$ close to canonical embedding of v.

1: $l, r := i,\ j$
2: lift_candidate $:= \mathcal{V}^{-1}_{d''/d'}(0, 0, \ldots, \overbrace{\left(v \cdot y^{(l+r)//\kappa}\right)}^{\text{at position } p}, \ldots, 0)$, where $p = (l + r) \mod \kappa$
3: $u := \text{NearestPlanes}(v, \mathcal{L}'_{i//\kappa,j//\kappa},)$
4: **return** u

Докажем корректность работы этой процедуры. Пусть мы применили оператор матрификации к элементу $u \in \mathcal{L}_{i//\kappa,j//\kappa}$ из $K_{d''}$, записывающемуся над подполем $K_{d'}$ как $u = (u_0, \ldots, u_{\kappa-1})$. В итоге мы получим матрицу:

$$B' = \begin{pmatrix} u_0 & u_1 & \ldots & \ldots & \ldots & u_{\kappa-1} \\ yu_{\kappa-1} & u_0 & \ldots & \ldots & \ldots & u_{\kappa-2} \\ \ddots & \ddots & \ldots & \ldots & \ddots & \ddots \\ yu_{\kappa-l} & yu_{\kappa-l-1} & \ldots & y^{(l+r)//\kappa}u_{l,r} & \ldots & u_{\kappa-1-l} \\ \ddots & \ddots & \ldots & \ldots & \ddots & \ddots \\ yu_1 & yu_2 & \ldots & \ldots & \ldots & u_0 \end{pmatrix}, \tag{14}$$

где $y = x^\kappa$ - основание системы образующих $\mathcal{O}_{K_{d''}}$.

l-я строка этой матрицы есть $\mathcal{V}_{d''/d'}(x^l u)$. Алгоритм Lift получает на вход элемент (пусть для начала для конкретики это будет $u_{l,r}$), а также индексы l и r, запоминающие, к какой из подрешёток он принадлежит. Для того, чтобы получить элемент v, принадлежащий решётке, порождённой u над $K_{d''}$, такой, что $\mathcal{V}_{d''/d'}(x^l v) = u_{l,r}$ нам нужно:

1. Составить вспомогательный вектор $w = (0, 0, \ldots, \overbrace{\left(u_{l,r} \cdot y^{(l+r)//\kappa}\right)}^{\text{at position } p}, \ldots, 0)$ как будто, все координаты (относительно базиса решётки) исходного вектора u, кроме координаты под номером $l + r \mod \kappa$ были нулевыми.
2. Девекторизовать w, получив $v = \mathcal{V}^{-1}_{d''/d'}(w)$

Переписав последнее, получим:

$$v = \mathcal{V}^{-1}_{d''/d'}((0, 0, \ldots, \overbrace{(u_{l,r} \cdot y^{(l+r)//\kappa})}^{\text{at position } p}, \ldots, 0)). \tag{15}$$

Этот вектор может не лежать в решётке $u \in \mathcal{L}_{i//\kappa, j//\kappa}$, но мы можем вызвать Nearest Planes алгоритм, чтобы он принадлежал.

Таким образом, ввиду того, что для $d' = 1, d'' = p$ для простого p Lift работает тривиально (мы просто берём в качестве целевого вектора нулевой и заменяем его $(l + r) \mod p$-ю координату на r-ю координату l-го базисного вектора, и в получившейся решётки размерности 1 над $\mathbb{Q}$ ищем кратчайший ненулевой вектор [то есть число ± 1]), то по индукции получаем, что мы всегда можем получить вектор из исходной решётки.

Если же мы вызвали SVP-оракул и он нам вместо $u_{l,r}$ вернул более короткий элемент из решётки, порождённой им, то мы можем расписать его как $\ell + u_{l,r}$, где $\ell \in K_{d'}$ само принадлежит решётке, порождённой $u_{l,r}$ (это гарантируется SVP-оракулом). Так как для векторов над $K_{d''}$ элемент $\ell \in K_{d'} \subset K_{d''}$ является скаляром, то мы можем заключить, что:

$$v = \mathcal{V}^{-1}_{d''/d'}((0, 0, \ldots, \overbrace{(u_{l,r} \cdot y^{(l+r)//\kappa})}^{\text{at position } p}, \ldots, 0)) + \mathcal{V}^{-1}_{d''/d'}((0, 0, \ldots, \overbrace{(\ell \cdot y^{(l+r)//\kappa})}^{\text{at position } p}, \ldots, 0)), \tag{16}$$

что аналогично можно поднять до вектора из $\mathcal{L}_{i//\kappa, j//\kappa}$ при помощи Nearest Planes алгоритма.

Функция $\mathbb{R}$-NearestPlanes является аналогом алгоритма Babai Nearest Plane. Однако, координата, возвращаемая на j-й итерации цикла оригинального алгоритма заменяется на C_j ближайших к ней чисел (включая её саму) и на следующую итерацию подаётся уже не один вектор, а несколько. А именно в C_j раз больше, чем было в начале итерации. Эта функция необходима для определения её аналога для случая алгебраической решётки, представленной в домене частоты.

Algorithm 6 $\mathbb{R}$-NearestPlanes

Input: $t \in \mathbb{R}^n$, matrix L from LDL^* decomposition of $B \cdot B^*$, $C \in (\mathbb{Z}^+)^n$.
Output: vector $u \in \mathcal{L}(B)$ such that $(u-t) \cdot B \in \mathcal{P}(\tilde{B})$

1: $Z := \{\vec{\mathbf{0}}\}$
2: **for** $j = n-1, \cdots, 0$ **do**
3: $\quad \overline{T} := \{t_j + \sum_{j<i<n}(t_i - z_i^{(k)}) \cdot L_{i,j}\}$ for all $k = \prod_{i=n-j+1}^{n-1} C_i$
4: $\quad Z := \{z^{(k)}\} = \bigcup_{0 \leqslant k < \prod_{i=n-j+1}^{n-1} C_i} \left(\{\ell \cdot \lfloor t^{(k)} \rceil\}_{\lfloor -C_j/2 \rceil < \ell \leqslant \lfloor C_j/2 \rceil}\right)$
5: $\quad$ **Exclude** from Z vectors that are too far away from t;
6: **end for**
7: **return** Z

В случае с модульными решётками вызвать $\mathbb{R}$-NearestPlanes напрямую не представляется возможным - операция нахождения ближайших "целых" нетривиальна. Для этого мы переводим алгоритм 4 из [DP16] в NearestPlanes-аналог, описанный в [LP10].

Algorithm 7 FastFourierNearestPlanes

Input: $t \in \mathcal{Z}_d^n$, a precomputed tree **Tree**, $C \in (\mathbb{Z}^+)^{d'}$.
Output:

1: $Z := \{\vec{\mathbf{0}}\}$
2: **for** $j = n-1, \cdots, 0$ **do**
3: $\quad \overline{T} := \{t_j + \sum_{j<i<n}(t_i - z_i^{(k)}) \cdot L_{i,j}\}$ for all $k = \prod_{i=n-j+1}^{n-1} C_i$
4: $\quad Z := \{z^{(k)}\} = \bigcup_{0 \leqslant k < \prod_{i=n-j+1}^{n-1} C_i} \left(\mathcal{V}_{d''/d}^{-1}\left(\mathbb{R}\text{-NearestPlanes}\left(\mathcal{V}_{d''/d}(t_j^{(k)}), \mathbf{Tree}_j, \mathcal{V}_{d''/d}(C)_j\right)\right)\right)$
5: $\quad$ **Exclude** from Z vectors that are too far away from t;
6: **end for**
7: **return** Z

5 Анализ алгоритма

5.1 Качество

Пусть для данного псевдо-LLL редуцированного базиса B ранга k над K_d и размера блока β мы получили $k\left(\frac{d}{\beta}\right)^2$ списков из $2^{\Theta(\beta)}$ векторов над $\mathcal{O}_{K_\beta}$, среди которых есть вектор длины не превышающей $\lambda_1(\mathcal{L})$. Также предположим, что d - логарифмически гладкое число. Тогда этот вектор будет поднят не более, чем $1 + \log(d) - \log\beta$ раз. При каждом подъёме его длина будет зависеть от двух факторов - насколько большой будет ошибка, привнесённая алгоритмом NearestPlanes и насколько она компенсируется слиянием списков.

Ошибка при подъёме зависит от длины векторов Грама-Шмидта. Пусть базис решётки $\mathcal{L}$ является псевдо-LLL редуцированным (редуцированным при помощи алгоритма [KEF19]).

Пусть при слиянии списков длина векторов уменьшается в $\delta > 1$ раз. Вектор $v \in K_d'^k$ является проекцией какого-то, скорее всего, достаточно короткого вектора. Поэтому можно ввести некое понятие "правильности" декодирования. j-я координата вектора v отгадана "правильно" если существует достаточно короткий вектор, проекция которого на первые j координат совпадает с v. Тогда, положив $C_j := ||b_j^*|| / \min(b_i^*)$ с большой вероятностью будем получать правильные координаты [4]. Однако, брать слишком большие C_j тоже неприемлемо - их произведение равно количеству векторов, которые

[4] Формализовать!

нужно хранить. В таком случае положим, что $C_j = O(1)$ для локальных параметров k и d. В случае "неправильного"угадывания j-й координаты квадрат нормы увеличивается как $||v^{(j)}||^2 = ||v^{(j-1)}||^2 + ||b_j^*||^2$, где верхний индекс $j-1$ в скобках указывает на v до присоединения координаты, а верхний индекс j - после. Таким образом, вероятность угадывания "правильной" координаты растёт с ростом C_j [5] и это позволяет угадывать больше "правильных" координат.

Отметим, что $\tilde{\boldsymbol{B}}_j$ в обозначениях следствия **??** следует воспринимать как аналоги векторов Грама-Шимдта, только для алгебраических решёток ранга k над полем степени d. Благодаря оператору матрификации от числа d можно "откалывать" делители и переносить их в значение ранга k. Поэтому имеет смысл рассматривать значения $d'_i, 0 \leqslant \ldots \leqslant d \leqslant \ldots \leqslant dk$ как бы интерполируя профиль $||\tilde{\boldsymbol{B}}_j||$ для каждого значения d', которое делит d. В таком случае получаем, что, если вызывать KEF-LLL над полем степени 1 (то есть над $\mathbb{Q}$), то отношение соседних длин векторов Грама-Шмидта будет равно $O(1)$, что соответствует оригинальному условию редукции по размеру. Если же в качестве d' выбрать $O(\log^{O(1)} d)$, то придём к отношению $2^{O(\log^{O(1)}(d))}$. Из этих рассуждений следует, что для профиля с хорошими характеристики нам следует уменьшать степень поля.

Пусть степень поля выбрана как $d' = O(\log^{O(1)} d)$, где d - степень исходного числового поля и базис $\boldsymbol{B}_j, 0 \leqslant j < k$ - псевдо-LLL редуцирован. [DP16] гарантирует, что Nearest Plane алгоритм рассматривает Грам-Шмидт профиль базиса алгебраического базиса, как классический Грам-Шмидт профиль $\tilde{\boldsymbol{B}}_i, 0 \leqslant i < kd$ базиса $\boldsymbol{B'}_j$, полученного d' ротациями каждого из векторов. В таком случае предположим, что проекции ротаций таких векторов на векторное пространство, порождённое последующими Грам-Шмидт векторами не сильно короче таких векторов.

Из свойств скорости убывания Грам-Шмидт профиля LLL-редуцированного базиса (которое также ограничено выполняется для псевдо-LLL) и того факта, что i-е координаты кратчайшего вектора (относительно базиса решётки) ограничены по модулю как $2\lambda(\mathcal{L})/||\tilde{B}'_i||$, следует, что, выбрав C_i как $2\lambda(\mathcal{L})/||\tilde{B}'_i||$, мы отыщем ближайший к данному вектор.

Суммируя всё вышеперечисленное, при "правильном" отгадывании всех координат приходим к длине вектора ошибок при работе Nearest Planes ограниченной сверху, как:

$$qkd \cdot \sum_{i=0}^{kd} \frac{2\lambda(\mathcal{L}) \cdot C_i}{||\tilde{B}'_i||} \leqslant O(qk^2d^2) \cdot \lambda(\mathcal{L}), \tag{17}$$

в предположении, что координаты векторов базиса $\boldsymbol{B}$ расположены равномерно в интервале $(-q, q)$, как это происходит, например, в NTRU.

Заметим, что вышеприведённые рассуждения рушатся как только мы "неправильно" отгадываем координату, так как получившийся вектор уже может не являться проекцией некоторого короткого вектора решётки. Так как профиль Грама-Шмидта особо мал для i приближающихся к kd, то последние C_i стоит выбирать достаточно большими - чтобы ошибки при работе алгоритма начали накапливаться не с самого начала (алгоритм работает с координатами, начиная с $kd-1$ по 0), а несколько позже. Практика показывает, что как только координата "правильного" вектора была отгадана неверно, дальнейшие ошибки начинают быстро накапливаться, увеличивая длину вектора.

[5] Но я и понятия не имею как

Назовём C_i "хорошим" , если $C_i = \lceil 2\lambda(\mathcal{L})/||\tilde{B}'_i||\rceil$. Пусть $0 \leqslant i' < kd$ - индекс, начиная с которого нами были выбраны все "хорошие" C_i. Тогда можно полагать, что на шаге $kd - i'$ работы Nearest Planes (он идёт от последних векторов к первым) мы "неправильно" отгадаем координату и начнёт накапливаться ошибка. В таком случае свойства LLL-редуцированного базиса утверждают, что фактор апроксимации при решении задачи CVP будет равен:

$$2^{2i'}||\tilde{B}'_{kd}||/4. \tag{18}$$

Выбрав $i' = kd/\log^{O(1)} kd$, мы с одной стороны получим "мягко" субэкспоненциальный фактор апроксимации $2^{2kd/\log^{O(1)} kd}||\tilde{B}'_{kd}||/4$, при этом сам алгоритм Nearest Planes будет работать за время $2^{O(kd\cdot(1-1/\log^{O(1)}(kd)))}$.

Повторить подъём нам придётся примерно $\log(kd/\beta)$ раз, поэтому итоговым аппроксимационным фактором будет:

$$2^{\frac{2kd}{\log^{O(1)}}kd\cdot\log(kd/\beta)}||\tilde{B}'_{kd}||^{\log(kd/\beta)}/4. \tag{19}$$

При этом, если степень выражения $\log^{O(1)}$ больше или равна 1. То мы можем упростить это выражение как:

$$2^{\frac{2kd}{\log^{O(1)} kd}}||\tilde{B}'_{kd}||^{\log(kd/\beta)}/4 \tag{20}$$

5.2 Время работы

В алгоритме 1 преобладает экспоненциальная сложность шагов 10-13, а именно $k\left(\frac{d}{\beta}\right)^2 \cdot 2^{2\Theta(\beta)+O(1)}$. Перед 16 шагом алгоритма имеем $\left(\frac{kd}{\beta}\right)^2 2^{\Theta(\beta)}$ векторов. Каждый будет участвовать в слиянии и подъёме. Для каждого такого вектора самым затратным по времени будет последний подъём в исходную решётку (остальные спрячутся за O-нотацией). Как было выведено в предыдущем пункте, при выборе индекса $i' = kd/\log^{O(1)} kd$, до которого все $C_i = 0$, и после которого все C_i - "хорошие" каждому вектору соответствует время подъёма $2^{O(kd\cdot(1-1/\log^{O(1)}(kd)))}$. Перемножив количество векторов на время подъёма каждого, получим общее время работы Nearest Planes:

$$\left(\frac{kd}{\beta}\right)^2 2^{\Theta(\beta)} \cdot 2^{O\left(kd\cdot\left(1-\frac{1}{\log^{O(1)} kd}\right)\right)}. \tag{21}$$

Список литературы

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC'16, page 191–198, 2016.

[DWZ18] Wei Dai, William Whyte, and Zhenfei Zhang. Optimizing polynomial convolution for ntruencrypt. Cryptology ePrint Archive, Report 2018/229, 2018. `https://ia.cr/2018/229`.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer, 1998.

[KEF19] Paul Kirchner, Thomas Espitau, and Pierre-Alain Fouque. Algebraic and euclidean lattices: Optimal lattice reduction and beyond. Cryptology ePrint Archive, Report 2019/1436, 2019. https://ia.cr/2019/1436.

[LP10] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. Cryptology ePrint Archive, Report 2010/613, 2010. https://ia.cr/2010/613.

[May99] Alexander May. Cryptanalysis of NTRU. *preprint, February*, 1999.

[PMHS19] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-svp in ideal lattices with pre-processing. Cryptology ePrint Archive, Report 2019/215, 2019. https://ia.cr/2019/215.