Lattie Sieving 1

Lattice Sieving Algorithms and Their Implementation

Wenling Liu, Bohang Chen, Shanghai Jiao Tong University.

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CRYPTO-SYSTEM LABORATORY, SJTU

Table of Contents

Introduction

Introduction

In this talk, we talk about sieving algorithm for solving (approximation) SVP on lattice.

Outline

- **Introduction**
  This it is. The outline, background.
- **Basics**
  Review the basics of lattice, GSO, and basis reduction.
- **The Sieving Algorithms**
  The core of sieving algorithms, the intuition, and its evolution
- **The Sieving Tricks**
  Algorithms beyond sievers that accelerates sieve
- **G6K Implementation**
  The overview of sieving library G6K that implements all above
- **Future Plan**
  Our plan on sieving research.

SVP Solvers

SVP is a **NP**-hard lattice problem that can be solved by following exponential complex algorithms:

- Enumeration (super exponential time, polynomial memory)
- Sieving (exponential time, exponential memory)
- Basis Reduction (often heuristic time, often only solves uSVP)

Enumeration was once faster than sieving in practice, but sieving is now faster in both theoretical and practical.

Main Sieving Breakthroughs

- [NV08] shows that sieving is practical, and solved for dim 50
- [LM18] The totally heuristic progressive sieve speeds up several algorithms for more than 20x!
- [ADH$^+$19] implements multithread sieving with serval tricks that solves 1.05-SVP for about 150 dims.
- [DSvW21] implements sieving on GPUs that solves 1.05-SVP for 180 dims, spending 51.6 days with 4 RTX 2080Ti and 1.5TB RAM.

Basics

Lattices, Basis, Volumn

Let $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_m \in \mathbb{R}^n$ and be linear independent. Let $\mathbf{B} = [\mathbf{b}_1\ \mathbf{b}_2\ \cdots\ \mathbf{b}_m]$, then we write

$$\Lambda = \mathcal{L}(\mathbf{B}) = \{ \sum_{i \in [m]} x_i \mathbf{b}_i | x_i \in \mathbb{Z} \}$$

We call $\mathbf{B}$ a **basis** of lattice Λ.
We define the volumn of $\mathcal{L}(\mathbf{B})$ to be $|\det \mathbf{B}|$.

Shortest Vector Problem

Shortest Vector Problem (SVP)

Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find a nonzero lattice vector $\mathbf{x} \in (\mathbf{B})$ s.t. $\|\mathbf{x}\| \leq \|\mathbf{y}\|$ for any other $\mathbf{y} \in \mathcal{L}(\mathbf{B})$.

Denote the vector length of SVP in Lattice $\mathcal{L}$ by $\lambda_1(\mathcal{L})$. We abuse notion $\lambda_1(\mathbf{B})$ for $\lambda_1(\mathcal{L}(\mathbf{B}))$.

γ-Approximate Shortest Vector Problem (SVP$_\gamma$)

Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, find a nonzero lattice vector $\mathbf{x} \in (\mathbf{B})$ s.t. $\|\mathbf{x}\| \leq \gamma(n) \cdot \lambda_1(\mathbf{B})$ for any other $\mathbf{y} \in \mathcal{L}(\mathbf{B})$.

We mainly focus on solving these 2 problems.

Gram-Schmidt Orthogonalization

For a lattice basis $\mathbf{B} = [\mathbf{b}_0, \cdots, \mathbf{b}_{n-1}] \in \mathbb{R}^{m \times n}$, the Gram-Schmidt orthogonal basis of $\mathbf{B}$ can be compute by

$$\begin{aligned} \mathbf{b}_0^* &= \mathbf{b}_0 \\ \mathbf{b}_1^* &= \mathbf{b}_1 - \mu_{1,0}\mathbf{b}_0^* \\ \mathbf{b}_{n-1}^* &= \mathbf{b}_{n-1} - \sum_{j=0}^{n-2} \mu_{n-1,j}(\mathbf{b}_j^*) \end{aligned}$$

where $\mu_{i,j} = \dfrac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$. We say $\mathbf{B}^* = [\mathbf{b}_0^*, \cdots, \mathbf{b}_{n-1}^*]$ the Gram-Schmidt Orthogonal Form of $\mathbf{B}$.

Other Notations

Let $\mathbf{v} \in \mathcal{L}(\mathbf{B})$, define

$$\pi_\ell(\mathbf{v}) := \mathbf{v} - \sum_{i=0}^{\ell-1} \frac{\langle \mathbf{b}_i^*, \mathbf{v} \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \mathbf{b}_i^*$$

It projects $\mathbf{v}$ to the perpendicular space of $\text{span}(\mathbf{b}_0, \cdots, \mathbf{b}_{\ell-1})$
We denote

$$\pi_\ell(\mathbf{B}) := [\pi_\ell(\mathbf{b}_\ell), \cdots, \pi_\ell(\mathbf{b}_{n-1})]$$

We abuse $\pi_\ell(\mathcal{L})$, where $\mathcal{L} = \mathcal{L}(\mathbf{B})$ for $\mathcal{L}(\pi_\ell(\mathbf{B}))$.
Denote

$$\mathbf{B}_{[\ell,r]} := [\pi_\ell(\mathbf{b}_\ell), \cdots, \pi_\ell(\mathbf{b}_r)]$$

And we denote $\mathbf{B}_{[\ell,n]}$ by $\mathbf{B}_\ell$.

Other Notations

$B_n(R)$: the ball with radius R and center $\mathbf{0}$ in $\mathbb{R}^n$.

Basis Reduction

We have following reduced conditions, $\mathbf{B}$ is

- size–reduced, if $|\mu_{i,j}| \leq 1/2$ for all $0 \leq j < i < n$.
- δ–LLL–reduced, if it is sized reduced and $(\delta - \mu_{i+1,i}^2)\|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^*\|^2$. $(1/4 < \delta \leq 1)$
- HKZ–reduced, if $\mathbf{b}_i^*$ is a shortest vector of $\pi_i(\mathcal{L})$ for all $1 \leq i < n$.
- β–BKZ–reduced, if sublattices $\mathcal{L}(\mathbf{B}_{[\ell,\ell+\beta]})$ is HKZ–reduced for all $0 \leq \beta < n - \ell$.

The Sievers

From Enumeration to Sieving

Enumerations solves SVP.

Let $S \subseteq \mathbb{R}^n$ be a subset containing $\lambda_1(\mathcal{L})$, retrieving all the vectors in $\mathcal{L} \cap S$ would give a vector of length $\lambda_1(\mathcal{L})$.

Sadly, the best such S contains $2^{O(n \log n)}$ vectors of $\mathcal{L}$ which leads to $2^{O(n \log n)}$ time complexity.

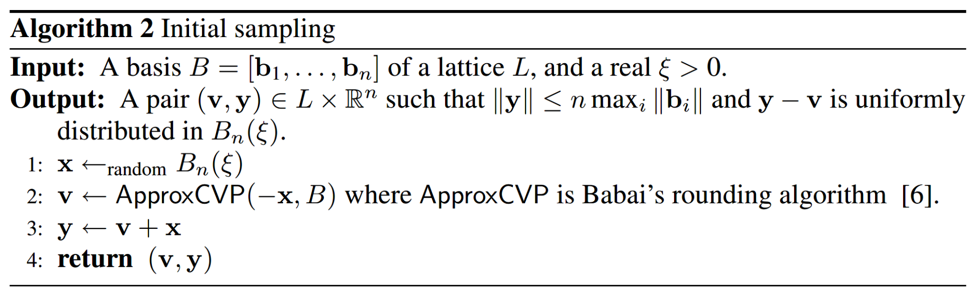
Sieving as Randomized Enumeration

Sieving algorithms choose smaller S, that $S \cap L$ is hard to be enumerated. However, SVP can be solved by random sampling. Choose $R = O(\lambda_1(\mathcal{L}))$ leads to $|\mathcal{L} \cap B_n(R)| = 2^{O(n)}$.

AKS Sieving

AKS sieving is proposed by Ajtai, Kumar and Sivakumar in [AKS01].
If we know how to uniformly sampling from such $\mathcal{L} \cap B_n(R)$, we are done. However, we don't.

Sampling from larger $\mathcal{L} \cap B_n(R_0)$ where $R_0 = 2^{O(n)}\lambda_1(\mathcal{L})$ is easy for us, so we sample from $\mathcal{L} \cap B_n(R_0)$ instead.



Algorithm 2 Initial sampling

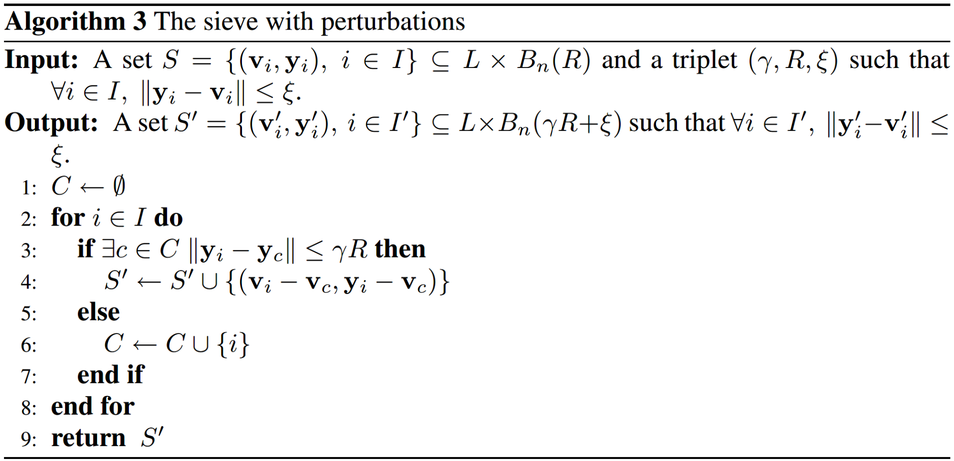
Input: A basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L, and a real $\xi > 0$.
Output: A pair $(\mathbf{v}, \mathbf{y}) \in L \times \mathbb{R}^n$ such that $\|\mathbf{y}\| \leq n \max_i \|\mathbf{b}_i\|$ and $\mathbf{y} - \mathbf{v}$ is uniformly distributed in $B_n(\xi)$.

```
1: $\mathbf{x} \leftarrow_{\text{random}} B_n(\xi)$
2: $\mathbf{v} \leftarrow \text{ApproxCVP}(-\mathbf{x}, B)$ where ApproxCVP is Babai's rounding algorithm [6].
3: $\mathbf{y} \leftarrow \mathbf{v} + \mathbf{x}$
4: return $(\mathbf{v}, \mathbf{y})$
```

Where ApproxCVP is Babai's rounding algorithm.

AKS Sieving

To obtain $\mathcal{L} \cap B_n(R)$ for $R = O(\lambda_1(\mathcal{L}))$, we "sieve" the database L. Namely, use shorter vector to substitute long vectors in L.



Algorithm 3 The sieve with perturbations

```
Input: A set $S = \{(\mathbf{v}_i, \mathbf{y}_i),\ i \in I\} \subseteq L \times B_n(R)$ and a triplet $(\gamma, R, \xi)$ such that
    $\forall i \in I,\ \|\mathbf{y}_i - \mathbf{v}_i\| \le \xi$.
Output: A set $S' = \{(\mathbf{v}'_i, \mathbf{y}'_i),\ i \in I'\} \subseteq L \times B_n(\gamma R + \xi)$ such that $\forall i \in I',\ \|\mathbf{y}'_i - \mathbf{v}'_i\| \le$
    $\xi$.
1: $C \leftarrow \emptyset$
2: for $i \in I$ do
3:     if $\exists c \in C\ \|\mathbf{y}_i - \mathbf{y}_c\| \le \gamma R$ then
4:         $S' \leftarrow S' \cup \{(\mathbf{v}_i - \mathbf{v}_c, \mathbf{y}_i - \mathbf{v}_c)\}$
5:     else
6:         $C \leftarrow C \cup \{i\}$
7:     end if
8: end for
9: return $S'$
```

AKS Sieving

In every round of sieving, we wish to decrease the database vector length by a factor of γ by paying $|C|$ vectors.

Some comments:

- We sieve the perturbated lattice vectors in sieving procedure, while recording their corresponding lattice vectors. This is necessary for correctness proof (which says we often obtain a nonzero shortest vector eventually).
- The sieving procedure works if we can bound λ_1 with some constants λ s.t. $\lambda < \lambda_1 < 1.01\lambda$. We have to try polynomial numbers of λ's if we don't know such range.

NV sieving introduce heuristics to overcome these performance drawbacks.

- Assume the vectors after sieving distributes uniformly.
- Introduce Gaussian heuristic to estimate the λ_1.

Gaussian Heuristic

Definition (Gaussian heuristic)

Let V be a measurable subset of $\mathbb{R}^n$, then $|V \cap \mathcal{L}|$ is approximately euqal to $\operatorname{vol}(V)/\operatorname{vol}(\mathcal{L})$.

We have

$$\operatorname{vol}(B_n(R)) = \frac{\pi^{n/2}}{\Gamma(n/2+1)} R^n \approx \frac{1}{\sqrt{n\pi}} \left(\frac{2\pi e}{n}\right)^{n/2} R^n$$

By letting it equal to $\operatorname{vol}(\mathcal{L})$ and solve for R, we obtain an approximation of λ_1,

$$\mathtt{gh}(\mathcal{L}) = \sqrt{n/2\pi e} \cdot \operatorname{vol}(\mathcal{L})^{1/n}$$

We denote the gaussian heuristic of n-dimensional lattice with volumn 1 by $\mathtt{gh}(n)$.

Gaussian Heuristic

Definition

Gaussian heuristic Let V be a measurable subset of $\mathbb{R}^n$, then $|V \cap \mathcal{L}|$ is approximately euqal to $\operatorname{vol}(V)/\operatorname{vol}(\mathcal{L})$.

We have

$$\operatorname{vol}(B_n(R)) = \frac{\pi^{n/2}}{\Gamma(n/2+1)} R^n \approx \frac{1}{\sqrt{n\pi}} \left(\frac{2\pi e}{n}\right)^{n/2} R^n$$

By letting it equal to $\operatorname{vol}(\mathcal{L})$ and solve for R, we obtain an approximation of λ_1,

$$\mathtt{gh}(n) = \sqrt{n/2\pi e} \cdot \operatorname{vol}(\mathcal{L})^{1/n}$$

The $\mathtt{gh}(n)$ works well for predicting the λ_1 of random lattice with dim larger than 50.

NV Sieving

NV sieve proposed by Nguyen and Vidick in [NV08] makes additional following assumption to simplify AKS sieve.

Definition (Heuristic)

Any stage after AKS sieve, the vectors in the database distributes uniformly in

$$C_n(\gamma, R) = \{\mathbf{x} \in \mathbb{R}^n : \gamma R \leq \|\mathbf{x}\| \leq R\}$$

This says we have negligible probability to lose vectors in database by collision until we achieve approximately λ_1-length vectors.

NV sieve doesn't promise an exact-SVP solution, but we often obtain such one in practice.

NV Sieving

NV Sieving choose a database of size $\sqrt{4/3}^n$, due to next lemma.

Lemma

Let $n \in \mathbb{N}$ and $2/3 < \gamma < 1$. Define $c_{\mathcal{H}} = 1/\left(\gamma\sqrt{1-\gamma^2/4}\right)$ and $N_C = c_{\mathcal{H}}^n \left\lceil 3\sqrt{2\pi}(n+1)^{3/2} \right\rceil$. Let N be an integer, and S a subset of $C_n(\gamma, R)$ of cardinality N whose points are picked independently at random with uniform distribution.

- *If $N_C < N < 2^n$, then for any subset $C \subseteq S$ of size at least N_C whose points are picked independently at random with uniform distribution, with overwhelming probability, for all $\mathbf{v} \in S$, there exists a $\mathbf{c} \in C$ such that $\|\mathbf{v} - \mathbf{c}\| \leq \gamma$*
- *If $N < 4\sqrt{\pi/2n}\sqrt{4/3}^n$, the expected number of points in S that are at distance at least γ from all the other points in S is at least $(1 - 1/n)N$.*

NV Sieving

From last lemma, we have the following:

- Choose gamma close to 1 allows us to have database of size $N = \sqrt{4/3+\epsilon}^n$, which gives running time of $O((4/3)^n)$.
- Database with size less than $N = \Omega(n^{-1/2}\sqrt{4/3}^n)$ is impossible since it losts $(1-1/n)N$ points in every round of sieving.

This force us to use database with size $O(\sqrt{4/3}^n) \approx O(2^{0.2075})$.

NV Sieve+

We do not decrease the database vector base size with a factor of γ.
We continuely consider pairs $\mathbf{v}_1, \mathbf{v}_2$ in the database. The NV Sieve+ algorithm `nv` works as follows:

- **Sieve**: Replace the longest vector in the database with $|\mathbf{v}_1 \pm \mathbf{v}_2|$ if $|\mathbf{v}_1 \pm \mathbf{v}_2| < \max\{|\mathbf{v}| : \mathbf{v} \in \mathtt{db}\}$.
- **Stop:** Stop when database is filled with $\sigma\sqrt{4/3}^n$ vectors in $B_n(\sqrt{4/3}\mathtt{gh}(\mathcal{L}))$.
- **Output:** Output the shortest $\mathbf{v}_1 \pm \mathbf{v}_2$ for $\mathbf{v}_1, \mathbf{v}_2 \in \mathtt{db}$.

σ is called the saturation ratio.

NV Sieve+

We obtain our general sieve algorithm from NV sieve+.

Algorithm 1: Lattice sieving algorithm.

```
Input  : A basis $\mathbf{B}$ of a lattice $\mathcal{L}$, list size $N$ and a saturation radius $R$.
Output: A list $L$ of short vectors saturating the ball of radius $R$.
1 Sample a list $L \subset \mathcal{L}$ of size $N$.
2 while $L$ does not saturate the ball of radius $R$ do
3     for every pair $\mathbf{v}, \mathbf{w} \in L$ do
4         if $\mathbf{v} - \mathbf{w} \notin L$ and $\|\mathbf{v} - \mathbf{w}\| < \max_{\mathbf{u} \in L} \|\mathbf{u}\|$ then
5             Replace a longest element of $L$ by $\mathbf{v} - \mathbf{w}$.
6 return $L$
```

Gaussian Sieve

Gaussian sieve is proposed by Micciancio and Voulgaris in [MV10]. It use Gaussian reduction in sieve.

`Gauss` divides the database in two parts:

- `Queue`: Always sorted, any $\mathbf{v}_1, \mathbf{v}_2 \in$ `Queue` are size-reduced.
- `List`: The rest part of `db`.

Every pairs $\mathbf{v}_1, \mathbf{v}_2 \in$ `Queue` is Gaussian reduced.

The operations, asymptotic running time, database size is similar to NV Sieve+, but works better in practice.

Bucketing

Checking pairs in `db` exhaustively is time-consuming. Thus we put vectors in `db` into buckets, and only check pairs in the same bucket. Notice that a vector can be put into multiple buckets.

- `bgj1`: Choose uniformly $\mathbf{d} \in \mathbb{R}^n$ s.t. $\|\mathbf{d}\| = 1$. An vector $\mathbf{v} \in$ `db` is put into $b_{\mathbf{d}}$ if $|\langle \mathbf{v}, \mathbf{d} \rangle| > \alpha \cdot |\mathbf{v}|$ for constant α. Time complexity $2^{0.349n+o(n)}$, space complexity $2^{0.2075n+o(n)}$.
- `bdgl`: Split the dimension n into dimension $n_1, \cdots, n_k$ s.t. $n_1 + \cdots + n_k = n$. Randomly choose centers C_k for each subdimension. Then the global bucket centers are $C = C_1 \times \pm C_2 \times \cdots \times \pm C_k$. For a vector $\mathbf{v}$, it is firstly bucketed into closets bucket centers and then put into gobal centers.

`bdgl` is faster asymptotically, while `bgj1` is practically faster. Both are implemented in G6K-GPU.

Tuple Sieve

Consider t-tuples $\mathbf{v}_1, \cdots, \mathbf{v}_t \in \mathtt{db}$ and replace longests with $\mathbf{v}_1 \pm \mathbf{v}_2 \pm \cdots \pm \mathbf{v}_t$ if these are shorter.

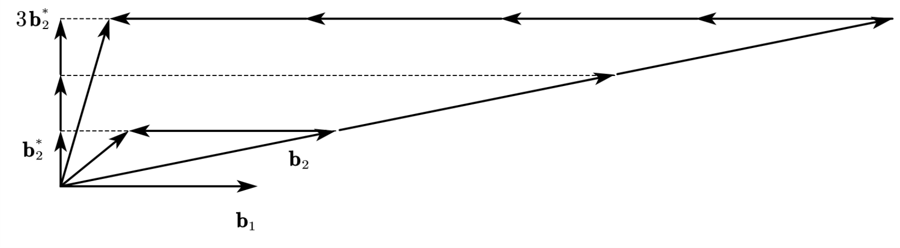
- More time-consuming but saves memory due to the good performance for smaller database.
- We set $t = 3$, i.e., use triple sieve, in practice.
- Also supports bucketing.
  For bucket with center $\mathbf{x}$ being a lattice vector, we test for $\mathbf{v}_1 \pm \mathbf{v}_2 \pm \mathbf{x}$ for each pair $\mathbf{v}_1, \mathbf{v}_2$ in the bucket. Time complexity $2^{O(0.396n+o(n))}$, space complexity $2^{O(0.1788n+O(n))}$.

The Sieving Tricks

Sublattice Vector Lifting

Let **w** be a vector from sublattic $\mathbf{B}_\ell$, how can we recover **x** s.t. $\pi_\ell(x) = \mathbf{w}$? This problem asks us to "lift" **w** back to **w** by undoing the projections.

There are uncountable legal lifts of **w**.



An arbitrary lift might be meaningless, we are committed to find short lifts.

Babai's Lift

Find the shortest lift for 1-dim is easy:

- Undo all the projections.
- Find the shortest lift by rounding.

We lift $\mathbf{w} = \mathbf{B}_\ell \mathbf{v}$ to $\mathbf{w}' = \mathbf{B}_{\ell-1}\mathbf{v}'$. Obviously, $\mathbf{w}_0 = \mathbf{B}_{\ell-1}(0, \mathbf{v})$ is a lift, but not the best. By lift $\mathbf{w}$ tp $\mathbf{w}_0$, we have add

$$\underbrace{\sum_{i=\ell}^{n-1} \mu_{i,\ell-1} v_i}_{c} \mathbf{b}^*_{\ell-1}$$

to $\mathbf{w}$. To add at least $\mathbf{b}^*_{\ell-1}$ as possible, we set $v_{\ell-1} = \lfloor c \rceil$ in v'. This allows us to add less than $1/2\ \mathbf{b}^*_{\ell-1}$ in $\mathbf{w}'$ to $\mathbf{w}$ and is the best.

Babai's Lift

To lift for mutiple dims, we repeat Babai's Lifting greedy. This seldom gives the best lift, but gives an well enough one.

Achieving the best lift for d-dim is hard. Which is equal to solve a CVP problem of dimensions d.

We next figure out when Babai's lift gives a best lift (for dim d). With this kind of lift, we solve SVP on $\mathcal{L}_d$ by sieving and lift its SVP to the SVP of $\mathcal{L}$. Then we have d **dimensions for free**.
We analysis dim for free with NV sieve, but it can be used else where.

Dim for Free

Let

$$L := \mathrm{NvSieve} = \{\mathbf{x} \in \mathcal{L}_d \setminus \{\mathbf{0}\} : \|\mathbf{x}\| \leq \sqrt{4/3} \cdot \mathtt{gh}(\mathcal{L}_d)\}$$

Let $\mathbf{s}$ be the SVP of $\mathcal{L}$. To have d-dims for free, we wish to have $\pi_d(\mathbf{s}) \in L$. Which gives

$$\mathtt{gh}(\mathcal{L}) \leq \sqrt{4/3} \cdot \mathtt{gh}(\mathcal{L}_d) \tag{1}$$

To lift $\pi_d(\mathbf{s})$ to $\mathbf{s}$ by Babai's lifting, a sufficient condition is $|\langle \mathbf{b}_i^*, \mathbf{s} \rangle| \leq \|\mathbf{b}_i^*\|^2$ for all $i < d$. This achieves by

$$\mathtt{gh}(\mathcal{L}) \leq \frac{1}{2} \min_{i<d} \|\mathbf{b}_i^*\| \tag{2}$$

which would not be a serious issue for small d in practice.

Dim for Free

To justify equation (1), we resort to BKZ-reduction.

Definition

Geometric Series Assumption Let $\mathbf{B}$ be a b-BKZ reduced basis of a lattice of volumn 1, the geometric assumption states that:

$$\|\mathbf{b}_i^*\| = \alpha_b^{\frac{n-1}{2}-i}$$

where $\alpha_b = \mathtt{gh}(b)^{2/b}$.

For b-BKZ basis $\mathbf{B}$ of $\mathcal{L}$, we have

$$\mathrm{vol}(\mathcal{L}_d) = \prod_{i=d}^{n-1} \|\mathbf{b}_i^*\| = \alpha_b^{d(d-n)/2}$$

which can be approximately rewrite as

$$d \ln \alpha_b \leq \ln(4/3) + \ln(1 - d/n) \tag{3}$$

Dim for Free

Take $b = n/2$ s.t. the time complexity of doing BKZ-reduction is negligible compared to such of sieving, we can achieve equation (3) with $d = \Theta(n/\ln n)$.

Lift On the Fly

To obtain more concrete dim for free, we lift every vector after in the sieving procedure, right after is has been obtained (even if it is not small enough for current sublattice basis). This is called "lift on the fly".

The concrete dim4free dimension number has to be obtain by doing experiment. In this case, BKZ-reduction is not necessary.

Progressive Sieve

Instead of work with dimension n directly, progressive sieve [LM18] increase dimension number gradually.

Progressive starts with sublattice of small dimension and its database, and iterates for the following until we achieve the goal dimension.

1. Do sieving to achieve good database under such dimension.
2. Increase sublattice dimension a little. Lift all the basis in `db` to new current sublattice vectors.
3. Sample new current sublattice vectors to `db`.

In implementation, we starts from $\mathcal{L}_{[n,n]}$, and iterate for $\mathcal{L}_{[n-1,n]}, \mathcal{L}_{[n-2,n]}, \cdots, \mathcal{L}_{[1,n]}$. Progressive sieve is totally heuristic, but works well in practice.

Flexible Insertion

During sublattice sieving, we insert good vector in the `db` back to basis **B** to improve it. The ideal comes from the sampling reduction algorithms.
We choose the order of insert by scoring.

Scoring. We keep insertion cadidates $\mathbf{c}_i$ (can be $\perp$ sometimes) for each position, we scoring them by

$$\varsigma(i) = \begin{cases} 0, & \text{if} \mathbf{c}_i = \perp \\ \theta^{-i} \cdot \parallel \mathbf{b}_i^* \parallel^2 / \parallel & \mathbf{c}_i \parallel^2, \text{otherwise} \end{cases}$$

and insert at position i with highest score first.

After insertion, the current sublattice basis will changed. Thus we have to recompute the vectors `db`.

Fast Pair/Triple Rejection - POPCNT/SimHash

Population count allows us to reject bad pairs and triples fast. We explain for pairs as an example. Since

$$\|\mathbf{v}_1 + \mathbf{v}_2\|^2 = \|\mathbf{v}_1\|^2 + \|\mathbf{v}_2\|^2 + 2\langle \mathbf{v}_1, \mathbf{v}_2 \rangle$$

We tend to reject $\mathbf{v}_1, \mathbf{v}_2$ with the same sign on the same component.

In implementation, we sample 256 random vectors $\mathbf{r}_1, \cdots, \mathbf{r}_{256}$ vectors, and store the signs of $\langle \mathbf{v}, \mathbf{r}_i$ for each $\mathbf{v} \in \mathtt{db}$. This is the SimHash of $\mathbf{v}$.

The counting of positives in SimHash$(\mathbf{v}_1) \oplus$ SimHash$(\mathbf{v}_2)$ can be done by a efficient modern CPU instruction POPCNT, which is provided by std::popcount in C++20. In the past, it has been provided by compilers.

A similar techinque can be used to do `bgj` bucketing.

BGJ15 Bucketing

SimHash-like bucketing `bgj15` :

- Sample a random vector $\mathbf{r}$ and let the vectors in db with similar SimHash to pass through the filter into the bucket.
- Repeat the filtering progress (with different SimHash setting) to get a even smaller bucket.
- Expecting to acquire $O(N)$ shorter vectors from db in $O(N)$ time, where N is the db size.

Fast Collision Dection

Store a hash $\texttt{uid} = H(\texttt{x})$ for every $\mathbf{x} \in \texttt{db}$. H has the property that $H(\texttt{x}_1) \pm H(\texttt{x}_2)$ is the $\texttt{uid}$ for $\mathbf{x}_1 \pm \mathbf{x}_2$.

Fast Lift Rejection - Dual Hash

[DSvW21] has introduced a new tech for fast rejecting vectors that non-likely to be worthy lifted called dual hash.

For context κ, ℓ, r, let $\mathcal{L} = \mathcal{L}_{[\ell,r]}$. To lift $k = \ell - \kappa$ dims, we choose a full row-rank matrix $\mathbf{D} \in \mathbb{R}^{h \times k}$ for $h \geq k$, with rows in $\mathcal{L}^*$ and define

$$\mathcal{H}_{\mathbf{D}} : \mathbf{t} \mapsto \mathbf{Dt}$$

It can be proved that

$$\mathrm{dist}(\mathcal{H}_{\mathbf{D}}(\mathbf{t}), \mathbb{Z}^h) \leq \sigma_1(\mathbf{D}) \cdot \mathrm{dist}(\mathbf{t}, \mathcal{L})$$

where $\sigma_1(\mathbf{D})$ is the largest singular value of $\mathbf{D}$. Dual Hash tech says we only lift such vectors that $\mathcal{H}_{\mathbf{D}}(\mathbf{t}) \leq H$ for some constant H.

Fast Lift Rejection - Dual Hash

The complexity of lifting is $O(k^2)$, the complexity of computing dual hash is $O(d \cdot k)$, which is larger than that of lifting is $d \geq k$.

We overcome this by precompute "dual hash helper" $\mathbf{Dt}$ for every $\mathbf{t}$. Due to its homomorphism property, the complexity can be reduce to $O(h)$ for each pair.

Section 5

G6K Implementation

Stateful Machine in G6K

G6K threat the sieving algorithm as a stateful machine, the full sieving algorithm is done by a series of instructions.

In G6K implementation, it stores the following states

- $\mathbf{B}$ the lattice basis, $\mathcal{B}^{\circ}$ the orthonormalized basis.
- κ, ℓ, r s.t. $0 \leq \kappa \leq \ell \leq r \leq d$, where $[\ell, r]$ is the sieving context, $[\kappa, r]$ is the lifting context.
- `db` the sieving database of vectors in $\mathcal{L}_{[\ell,r]}$
- $\mathbf{c}_{\kappa}, \cdots, \mathbf{c}_{\ell}$ the lifting candidates s.t. $\mathbf{c}_i \in \mathcal{L}_{[i,r]}$ or $\mathbf{c}_i = \perp$.

However, $\mathbf{B}_{[\ell,r]}$ is stored implicity due to its necessity in computation.

Database Structure

Database is stored by database `db` and compressed databased `cdb`. In `db` entry, we keep the following for a vector `e`:

- `e.x`: the vector coordinate in basis $\mathbf{B}_{[\ell,r]}$ ($\mathbf{v}$ on next page);
- `e.yr`: the vector coordinated under normalized basis $\mathbf{B}^{\circ}_{[\ell,r]}$ ($\mathbf{v}^{\circ}$ on next page);
- `e.cv`: the SimHash of `e`;
- `e.uid`: the hash of `e`;
- `e.len`: the length of `e`;
- `e.dual_helper`: the information for computing dual hash;
- `e.oft_helper`: information for computing lifting;

`cdb` is smaller and always being sorted. And every entry in `cdb` has an pointer to its pointed entry in `db`.

Instructions

The sieving context can be changed by following instructions.

- Extend Right ER: $\mathcal{L}_{[\ell,r]} \to \mathcal{L}_{[\ell,r+1]}$

$$(v_\ell, \dots v_r) \mapsto (v_0, \dots v_\ell, 0)$$
$$(v_\ell^\circ, \dots v_r^\circ) \mapsto (v_\ell^\circ, \dots v_r^\circ, 0)$$

- Shrink Left SL: $\mathcal{L}_{[\ell,r]} \to \mathcal{L}_{[\ell+1,r]}$

$$(v_\ell, \dots v_r) \mapsto (v_{\ell+1}, \dots v_r)$$
$$(v_\ell^\circ, \dots v_r^\circ) \mapsto \left(v_{\ell+1}^\circ, \dots v_r^\circ\right)$$

- Extend Left EL: $\mathcal{L}_{[\ell,r]} \to \mathcal{L}_{[\ell-1,r]}$

$$(v_\ell, \dots, v_r) \mapsto (-\lfloor c \rceil, v_\ell, \dots, v_r)$$
$$(v_\ell^\circ, \dots, v_r^\circ) \mapsto \left((c - \lfloor c \rceil) \cdot \left|\mathbf{b}_{\ell-1}^*\right|, v_\ell^\circ, \dots v_r^\circ\right)$$

Instructions

Their are some other instructions:

- $\texttt{Init}_{\mathbf{B}}$: Initialize the machine with basis $\mathbf{B}$.
- $\texttt{Reset}_{\kappa,\ell,r}$: Empty the database, set the context to (κ, ℓ, r).
- $\texttt{S}$: Sieve the database, lift the results in $\mathcal{L}_{[\kappa,r]}$. If we enable "on-the-fly lift", then all encountered vector is lifted to $\mathcal{L}_{[\kappa,r]}$.
- $\texttt{I}_i$, $\texttt{I}$: Insert candidate at position i, or insert to the position with highest scores. The sieving context goes to $[\ell + 1, r]$.
- $\texttt{Resize}_N$: Resize the database size to N, sample new vectors or remove longest to adapt new size.

Operation Examples

The plain sieve:

$$\texttt{Reset}_{[0,0,n-1]}, \texttt{S}, \texttt{I}_0$$

The progressive sieve:

$$\texttt{Reset}_{[0,0,0]}, (\texttt{ER}, \texttt{S})^n, \texttt{I}_0$$

Partial HKZ reduction with f dimensions for free (from left to right):

$$\texttt{Reset}_{0,f,f}, (\texttt{ER}, \texttt{S})^{n-f}, \texttt{I}_0, \texttt{I}_1, \cdots, \texttt{I}_{n-f-1}$$

Partial HKZ reduction with f dimensions for free (from right to left):

$$\texttt{Reset}_{0,d,d}, (\texttt{EL}, \texttt{S})^{n-f}, \texttt{I}_0, \texttt{I}_1, \cdots, \texttt{I}_{n-f-1}$$

Pump Operation

$\texttt{Pump}_{\kappa,f,\beta,s}$ sieves for dim β, and use f dim for free, extend from right to left, and always insert to the best position, s is the switch of down-sieving:

$$\texttt{Pump}_{\kappa,f,\beta,s} : \text{Reset}_{\kappa,\kappa+\beta,\kappa+\beta}, (\texttt{EL}, \texttt{S})^{\beta-f}, (\texttt{I}, \texttt{S}^{s})_{\beta-f}$$

- In exact-SVP, it is suggested to turn off s, i.e., $s = 1$.
- In GPU implementation that solves 1.05-SVP, $s = 1$.

Workout Operation

In practice, we don't know acurate dim for free (especially when enable on-the-fly lift), to obtain as much dim4free as possible, we test by `Workout`:

$$\texttt{Workout}_{\kappa,\beta,f,f^+,s} : \texttt{Pump}_{\kappa,\beta-f^+,\beta,s}, \texttt{Pump}_{\kappa,\beta-2f^+,\beta,s}, \texttt{Pump}_{\kappa,\beta-3f^+,\beta,s}, \cdots, \texttt{Pump}_{\kappa,\beta-f,\beta,s}$$

where f is the minimal dim4free. Notice that after every run of `Pump`, we obtain better basis.

GPU Sieve

Due to the memory bandwidth of GPU, we no longer insert the obtained vector back to database while sieving. The GPU sieving is sperated into 3 phases:

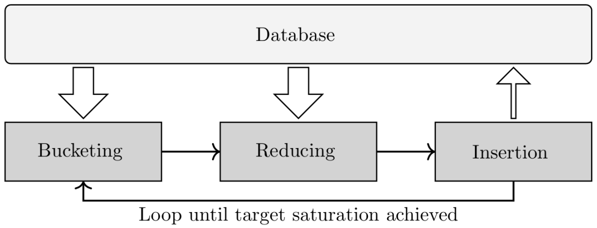


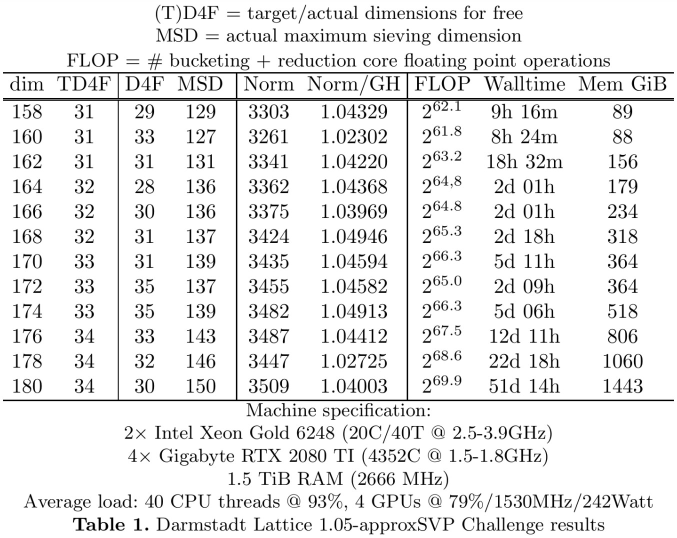
Fig. 2. High level diagram of the implemented Sieving process.

GPU Sieving

Compared to CPU, GPU siever has following characteristics:

- Only `x` representation is sent to GPU, other information (e.g., `yr` representation, length) is computed on the fly.
- Inner products of pairs $\mathbf{y}_1, \cdots, \mathbf{y}_s$ are computed together by $\mathbf{Y}^t\mathbf{Y}$ s.t. $\mathbf{Y} = [\mathbf{y}_1, \cdots, \mathbf{y}_s]$. This can be done very quickly by Tensor cores.
    - It seems SimHash is no longer used here.

GPU Sieving



(T)D4F = target/actual dimensions for free
MSD = actual maximum sieving dimension
FLOP = # bucketing + reduction core floating point operations

dim	TD4F	D4F	MSD	Norm	Norm/GH	FLOP	Walltime	Mem GiB
158	31	29	129	3303	1.04329	$2^{62.1}$	9h 16m	89
160	31	33	127	3261	1.02302	$2^{61.8}$	8h 24m	88
162	31	31	131	3341	1.04220	$2^{63.2}$	18h 32m	156
164	32	28	136	3362	1.04368	$2^{64,8}$	2d 01h	179
166	32	30	136	3375	1.03969	$2^{64.8}$	2d 01h	234
168	32	31	137	3424	1.04946	$2^{65.3}$	2d 18h	318
170	33	31	139	3435	1.04594	$2^{66.3}$	5d 11h	364
172	33	35	137	3455	1.04582	$2^{65.0}$	2d 09h	364
174	33	35	139	3482	1.04913	$2^{66.3}$	5d 06h	518
176	34	33	143	3487	1.04412	$2^{67.5}$	12d 11h	806
178	34	32	146	3447	1.02725	$2^{68.6}$	22d 18h	1060
180	34	30	150	3509	1.04003	$2^{69.9}$	51d 14h	1443

Machine specification:
2× Intel Xeon Gold 6248 (20C/40T @ 2.5-3.9GHz)
4× Gigabyte RTX 2080 TI (4352C @ 1.5-1.8GHz)
1.5 TiB RAM (2666 MHz)
Average load: 40 CPU threads @ 93%, 4 GPUs @ 79%/1530MHz/242Watt

Table 1. Darmstadt Lattice 1.05-approxSVP Challenge results

Future Plan

First Step

The G6K-GPU seems to be non-well-formed.

1. Rewrite the code with C++ and CUDA in a cmake project, based on our own modified well-formed C++ lattice algorithm library fplll.
2. Follow newest theoretical optimization.
3. Experiment with our own optimization with our own program.

However, the GPU siever seems to be well formed, we might test ideals merely involves siever by modifying the GPU siever.

Immatural Ideals

1. Tuple (> 3) sieve to save memory? For tuples $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_t]$, we check random $\mathbf{v}_a = \mathbf{Va}$ and $\mathbf{v}_b = \mathbf{Vb}$ with $\mathbf{a}, \mathbf{b}$ sampled uniformly from $\{-1, 0, 1\}^n$?
2. Can we enable SimHash in GPU setting? What about group by SimHash? Or can we bucket buckets into sub-buckets according to SimHash?

Bibliography

Bibliography I

Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens.
The general sieve kernel and new records in lattice reduction.
In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 717–746. Springer, 2019.

Miklós Ajtai, Ravi Kumar, and D. Sivakumar.
A sieve algorithm for the shortest lattice vector problem.
In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001.

Bibliography II

Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden.
Advanced lattice sieving on gpus, with tensor cores.
In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 249–279. Springer, 2021.

Thijs Laarhoven and Artur Mariano.
Progressive lattice sieving.
In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 292–311. Springer, 2018.

Bibliography III

Daniele Micciancio and Panagiotis Voulgaris.
Faster exponential time algorithms for the shortest vector problem.
In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1468–1480. SIAM, 2010.

Phong Q. Nguyen and Thomas Vidick.
Sieve algorithms for the shortest vector problem are practical.
J. Math. Cryptol., 2(2):181–207, 2008.