Sieving in practice: The Generalized Sieve Kernel (G6K)

Elena Kirshanova

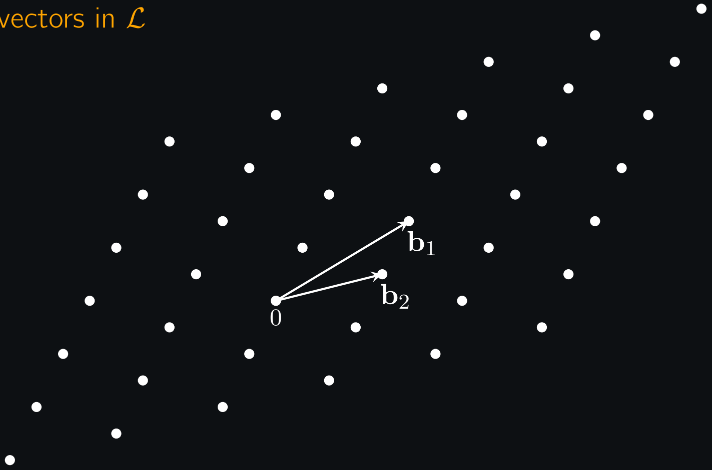
based on joint work with Martin R. Albrecht, Leo Ducas, Eamonn W. Postlethwaite, Gottfried Herold, Marc Stevens

The Simons Institute for the Theory of Computing
May 5, 2020

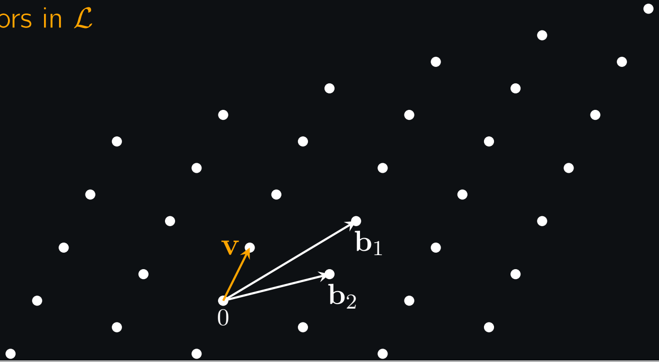
Part 0

Preliminaries

Short vectors in $\mathcal{L}$



Short vectors in $\mathcal{L}$



Given $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ – a basis of $\mathcal{L}$, the Shortest Vector Problem (SVP) asks to find non-zero $\mathbf{v}$ of minimal length.

We do not know $||\mathbf{v}_{\text{shortest}}||$ in general, but for any n-rank $\mathcal{L}$:

$$||\mathbf{v}_{\text{shortest}}|| \leq \sqrt{n} \cdot \det(\mathcal{L})^{1/n} \quad \text{(Minkowski's bound)}$$

Lattices of our interest

Goldstein-Mayer type of lattice with a basis given by columns:

$$B = \begin{pmatrix} p & x_1 & \dots & x_{n-1} \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix},$$

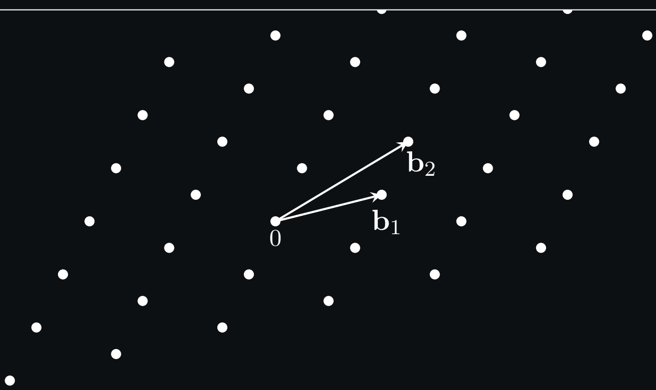
where p is a large prime and x_i are iid. uniform random from $\{0, \dots, p-1\}$.

$$\det(\mathcal{L}(B)) = p \implies ||\mathbf{v}_{\text{shortest}}|| \leq \sqrt{n} p^{1/n}$$

We'll be fine with $\mathbf{v}$ slightly longer than the shortest, e.g., $||\mathbf{v}|| \approx 1.05 \cdot \sqrt{n} \det(\mathcal{L})^{1/n}$ ($1.05-$ Hermite-SVP)

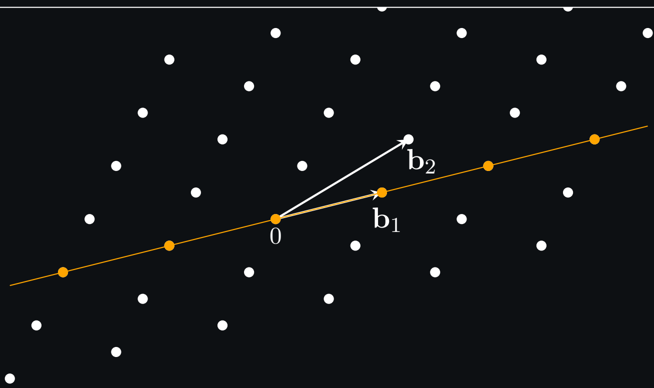
Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$



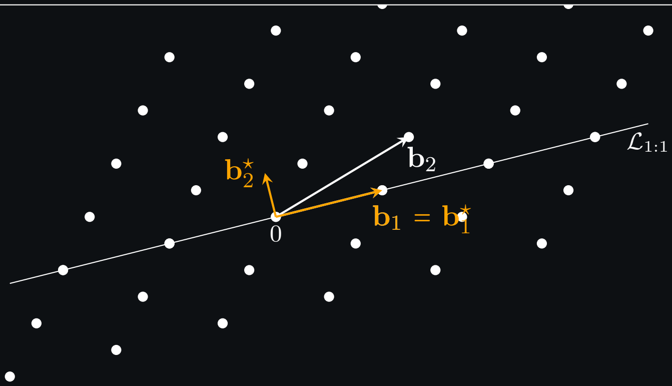
Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$



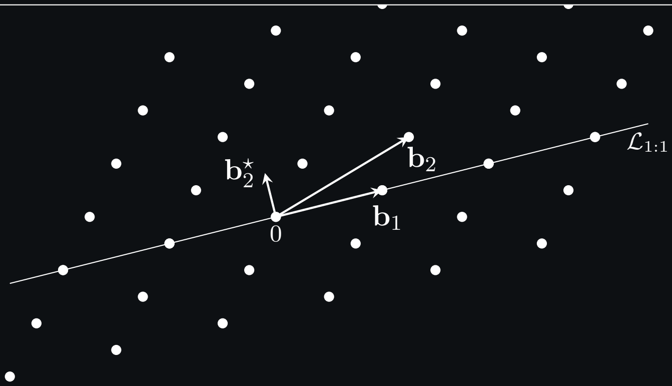
Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$
- $\mathbf{b}_i^\star$ is the projection of $\mathbf{b}_i$ on $\mathcal{L}_{1:i-1}^\perp$. (GSO)



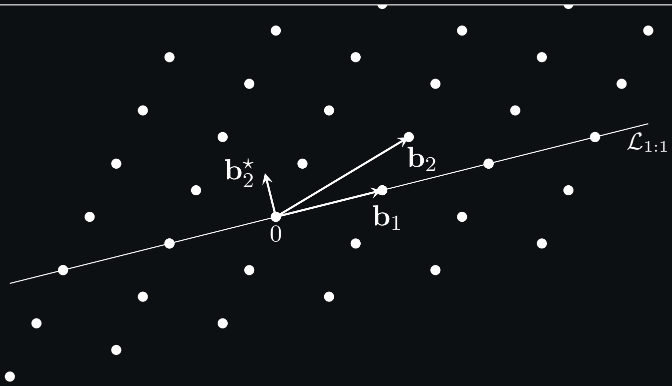
Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$
- $\mathbf{b}_i^\star$ is the projection of $\mathbf{b}_i$ on $\mathcal{L}_{1:i-1}^\perp$ (GSO)
- $\mathbf{b}_1, \mathbf{b}_2^\star, \ldots, \mathbf{b}_n^\star$- GSO basis of $\mathcal{L}$



Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$
- $\mathbf{b}_i^\star$ is the projection of $\mathbf{b}_i$ on $\mathcal{L}_{1:i-1}^\perp$ (GSO)
- $\mathbf{b}_1, \mathbf{b}_2^\star, \ldots, \mathbf{b}_n^\star$- GSO basis of $\mathcal{L}$
- often it is convenient to represent lattice vectors in GSO basis



What is inside g6k

The General Sieve Kernel implements

1. Exact-SVP
   The output is compared against the Gaussian heuristic
2. 1.05-Hermite SVP
   Darmstadt SVP-Challenge
3. BKZ
4. LWE
   Darmstadt LWE-Challenge

What is inside g6k

The General Sieve Kernel implements

1. Exact-SVP
   The output is compared against the Gaussian heuristic
2. 1.05-Hermite SVP
   Darmstadt SVP-Challenge
3. BKZ
4. LWE
   Darmstadt LWE-Challenge

Each of the above can use either of the following sieve:

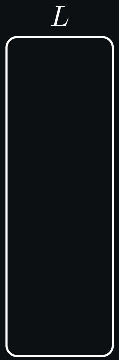
- `gauss_sieve`
- `nv_sieve` (Nguyen-Vidick sieve)
- `bjg1` (single- or multi-threaded) (Becker-Gama-Joux bucket sieve)
- `triple_sieve` (single- or multi-threaded)

Part I

nv_sieve

Nguyen-Vidick sieve

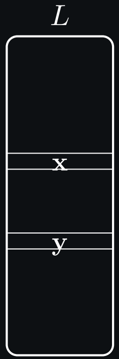
All sieving algorithms start by sampling lots of lattice vectors into a list L and by sorting it.



- Sampling can be done by 1. sampling the last coordinates wrt. GSO basis and 2. lifting to the full lattice using Babai

Nguyen-Vidick sieve

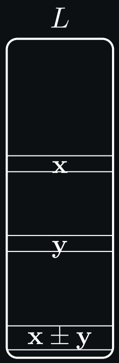
All sieving algorithms start by sampling lots of lattice vectors into a list L and by sorting it.



- Sampling can be done by 1. sampling the last coordinates wrt. GSO basis and 2. lifting to the full lattice using Babai
- Sieving searches for pairs $\mathbf{x}, \mathbf{y} \in L$ s.t. $\|\mathbf{x} \pm \mathbf{y}\|$ is small

Nguyen-Vidick sieve

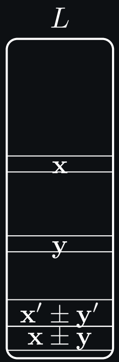
All sieving algorithms start by sampling lots of lattice vectors into a list L and by sorting it.



- Sampling can be done by 1. sampling the last coordinates wrt. GSO basis and 2. lifting to the full lattice using Babai
- Sieving searches for pairs $\mathbf{x}, \mathbf{y} \in L$ s.t. $\|\mathbf{x} \pm \mathbf{y}\|$ is small
- Once found replace the longest vector in L with $\mathbf{x} \pm \mathbf{y}$

Nguyen-Vidick sieve

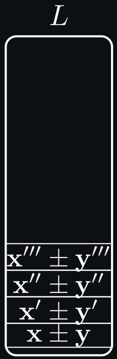
All sieving algorithms start by sampling lots of lattice vectors into a list L and by sorting it.



- Sampling can be done by 1. sampling the last coordinates wrt. GSO basis and 2. lifting to the full lattice using Babai
- Sieving searches for pairs $\mathbf{x}, \mathbf{y} \in L$ s.t. $\|\mathbf{x} \pm \mathbf{y}\|$ is small
- Once found replace the longest vector in L with $\mathbf{x} \pm \mathbf{y}$
- The next short $\mathbf{x}' \pm \mathbf{y}'$ replaces the current longest on L

Nguyen-Vidick sieve

All sieving algorithms start by sampling lots of lattice vectors into a list L and by sorting it.



- Sampling can be done by 1. sampling the last coordinates wrt. GSO basis and 2. lifting to the full lattice using Babai
- Sieving searches for pairs $\mathbf{x}, \mathbf{y} \in L$ s.t. $\|\mathbf{x} \pm \mathbf{y}\|$ is small
- Once found replace the longest vector in L with $\mathbf{x} \pm \mathbf{y}$
- The next short $\mathbf{x}' \pm \mathbf{y}'$ replaces the current longest on L
- Stop when “enough” short pairs are found

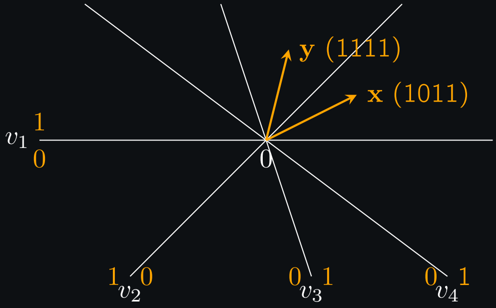
Runtime: $|L|^2$. For $|L| = \left(\frac{4}{3}\right)^{n/2}$, $T = 2^{0.415n}$

How to efficiently discard unpromising pairs, [Cha02, FBB+15, Duc18]

We spend most of the time testing if $\mathbf{x} \pm \mathbf{y}$ is short.
Need to compute the scalar product $|\langle \mathbf{x} , \mathbf{y} \rangle|$ fast
Most scalar products are useless, leading to no new vectors.

How to efficiently discard unpromising pairs, [Cha02, FBB+15, Duc18]

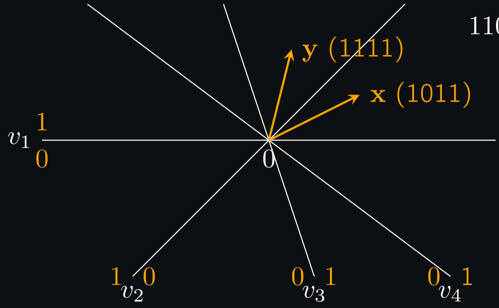
We spend most of the time testing if $\mathbf{x} \pm \mathbf{y}$ is short.
Need to compute the scalar product $|\langle \mathbf{x}, \mathbf{y} \rangle|$ fast
Most scalar products are useless, leading to no new vectors.



Close vectors are likely to lie in the same half-space.
Quick test with XOR + Popcount. We implement this test for all sieves.

How to efficiently discard unpromising pairs, [Cha02, FBB+15, Duc18]

We spend most of the time testing if $\mathbf{x} \pm \mathbf{y}$ is short.
Need to compute the scalar product $|\langle \mathbf{x}, \mathbf{y} \rangle|$ fast
Most scalar products are useless, leading to no new vectors.

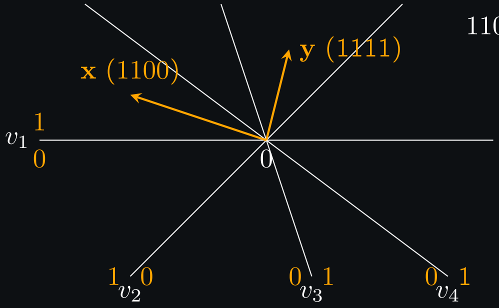


$$1101 \oplus 1111 = 0010$$

Close vectors are likely to lie in the same half-space.
Quick test with XOR + Popcount. We implement this test for all sieves.

How to efficiently discard unpromising pairs, [Cha02, FBB+15, Duc18]

We spend most of the time testing if $\mathbf{x} \pm \mathbf{y}$ is short.
Need to compute the scalar product $|\langle \mathbf{x}, \mathbf{y} \rangle|$ fast
Most scalar products are useless, leading to no new vectors.



$1100 \oplus 1111 = 0011$

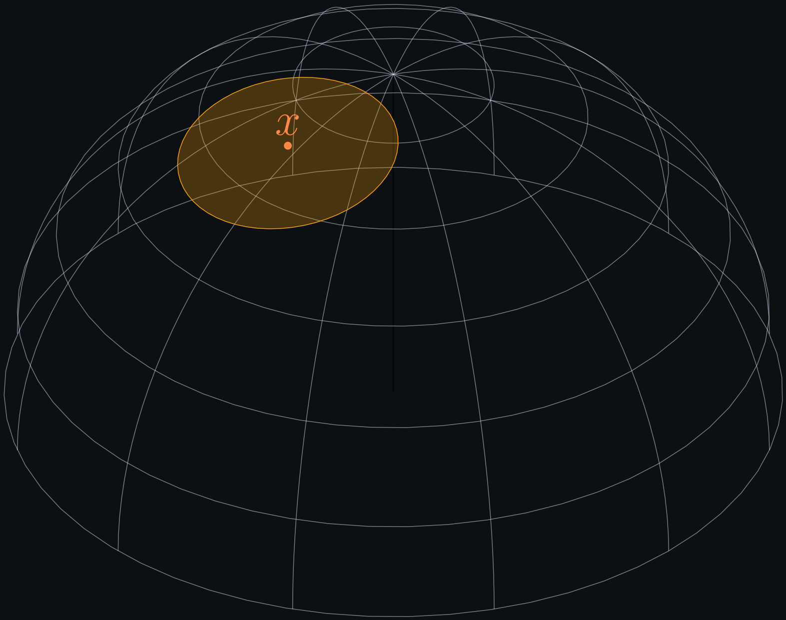
Close vectors are likely to lie in the same half-space.
Quick test with XOR + Popcount. We implement this test for all sieves.

Part II

bjg1

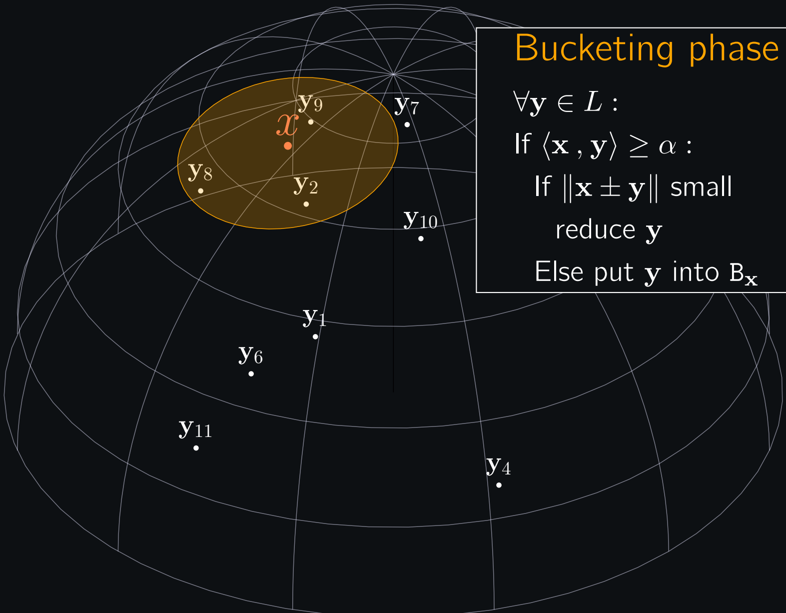
bjg1= NV Sieve + Buckets

Bucket center $\mathbf{x} \in L$ defines a region $\mathtt{B}_{\mathbf{x}}$



bjg1= NV Sieve + Buckets

Bucket center $\mathbf{x} \in L$ defines a region $\mathtt{B}_{\mathbf{x}}$



Bucketing phase

$\forall \mathbf{y} \in L :$

If $\langle \mathbf{x} \, , \mathbf{y} \rangle \geq \alpha :$

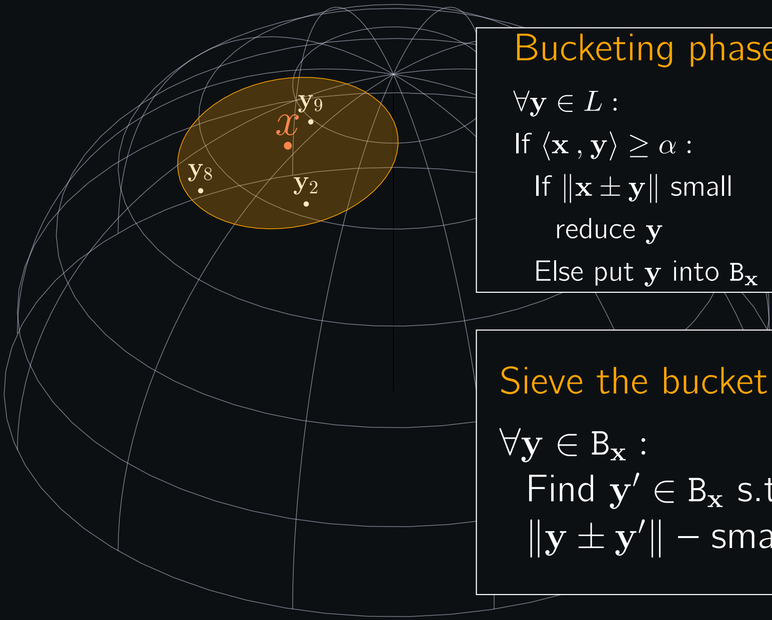
If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

bjg1= NV Sieve + Buckets

Bucket center $\mathbf{x} \in L$ defines a region $\mathtt{B}_\mathbf{x}$



Bucketing phase

$\forall \mathbf{y} \in L :$

If $\langle \mathbf{x} , \mathbf{y} \rangle \geq \alpha :$

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_\mathbf{x}$

Sieve the bucket

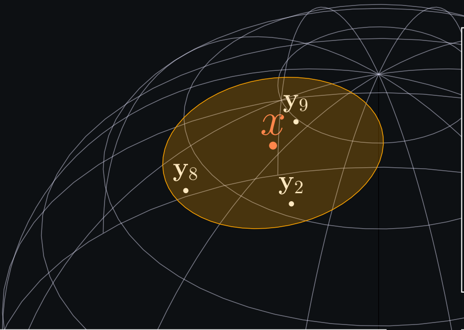
$\forall \mathbf{y} \in \mathtt{B}_\mathbf{x} :$

Find $\mathbf{y}' \in \mathtt{B}_\mathbf{x}$ s.t.

$\|\mathbf{y} \pm \mathbf{y}'\|$ – small

bjg1= NV Sieve + Buckets

Bucket center $\mathbf{x} \in L$ defines a region $\mathtt{B}_{\mathbf{x}}$



Bucketing phase

$\forall \mathbf{y} \in L$:

If $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha$:

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

bjg1 strategy:

choose $\mathbf{x}$ randomly from L

for $2^{0.142n}$ centres to find all pairs:

$T = 2^{0.349n}$, $M = 2^{0.2075n}$

Sieve the bucket

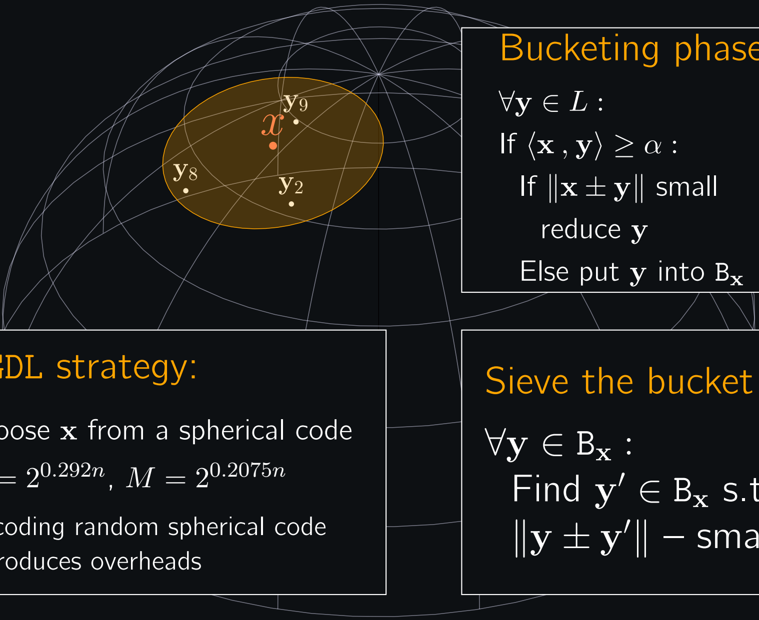
$\forall \mathbf{y} \in \mathtt{B}_{\mathbf{x}}$:

Find $\mathbf{y}' \in \mathtt{B}_{\mathbf{x}}$ s.t.

$\|\mathbf{y} \pm \mathbf{y}'\|$ – small

bjg1= NV Sieve + Buckets

Bucket center $\mathbf{x} \in L$ defines a region $\mathtt{B}_\mathbf{x}$



Bucketing phase

$\forall \mathbf{y} \in L$:

If $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha$:

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_\mathbf{x}$

BGDL strategy:

choose $\mathbf{x}$ from a spherical code

$T = 2^{0.292n}$, $M = 2^{0.2075n}$

decoding random spherical code introduces overheads

Sieve the bucket

$\forall \mathbf{y} \in \mathtt{B}_\mathbf{x}$:

Find $\mathbf{y}' \in \mathtt{B}_\mathbf{x}$ s.t.

$\|\mathbf{y} \pm \mathbf{y}'\|$ – small

Parallelized `bjg1`

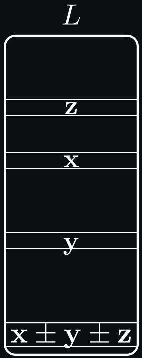
- Parallelization is done by having different threads work with different buckets.
- Reading the database of vectors is lock free
- Insertions of new shorter vectors into the global list are delayed and are executed in batches
- Sorting is complicated.

Part III

triple_sieve

Triple sieve

Motivation: reduce $2^{0.2075n}$ memory

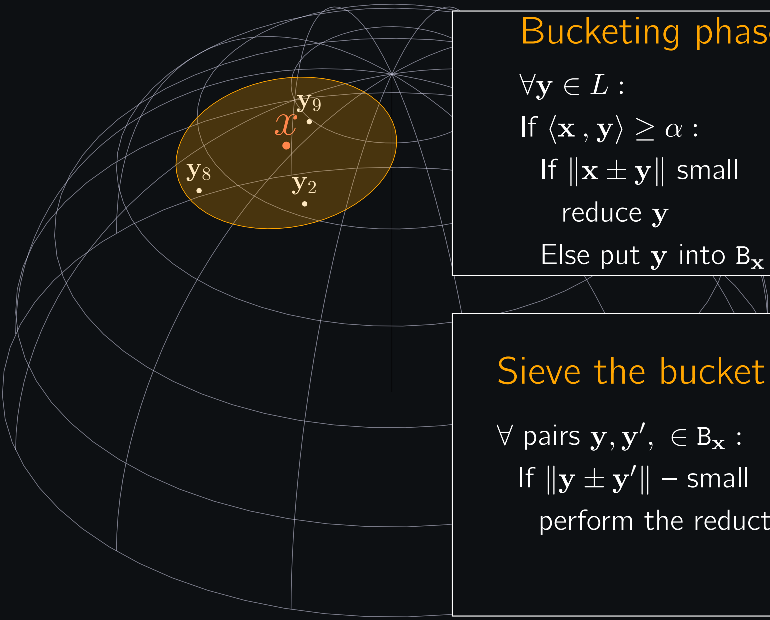


- 3-Sieve searches for triples $\mathbf{x}, \mathbf{y}, \mathbf{z} \in L$ s.t. $\|\mathbf{x} \pm \mathbf{y} \pm \mathbf{z}\|$ is small
- Once found replace the longest vector in L with $\mathbf{x} \pm \mathbf{y} \pm \mathbf{z}$
- Stop when "enough" short pairs are found

Memory optimal regime: $M = 2^{0.1887n}$, $T = 2^{0.3588n}$

Generalises to k-Sieve but taking $k > 3$ seems impractical

bgj1



Bucketing phase

$\forall \mathbf{y} \in L$:

If $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha$:

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

Sieve the bucket

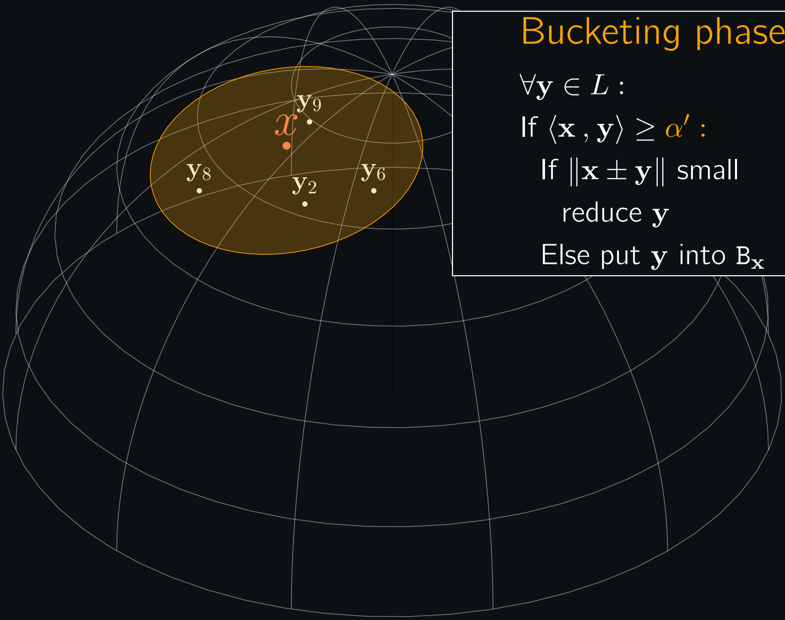
$\forall$ pairs $\mathbf{y}, \mathbf{y}',\ \in \mathtt{B}_{\mathbf{x}}$:

If $\|\mathbf{y} \pm \mathbf{y}'\|$ – small

perform the reduction

`triple_sieve` vs. `bjg1`

`triple_sieve`



Bucketing phase

$\forall \mathbf{y} \in L$:

If $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha'$:

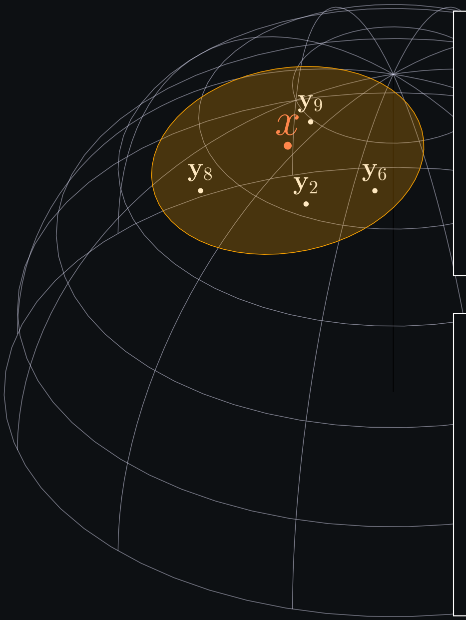
If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

triple_sieve vs. bjg1

triple_sieve



Bucketing phase

$\forall \mathbf{y} \in L$:

If $\langle \mathbf{x} \, , \mathbf{y} \rangle \geq \alpha'$:

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

Sieve the bucket

$\forall$ pairs $\mathbf{y}, \mathbf{y}',\ \in \mathtt{B}_{\mathbf{x}}$:

If $\|\mathbf{y} \pm \mathbf{y}'\|$ – small

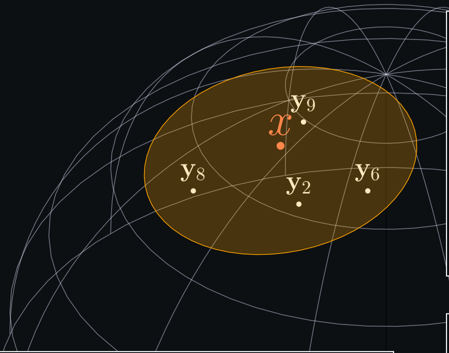
perform the reduction

Else if $\|\mathbf{x} \pm \mathbf{y} \pm \mathbf{y}'\|$ – small

perform the reduction

`triple_sieve` vs. `bjg1`

`triple_sieve`



Bucketing phase

$\forall \mathbf{y} \in L:$

If $\langle \mathbf{x}, \mathbf{y} \rangle \geq \alpha'$:

If $\|\mathbf{x} \pm \mathbf{y}\|$ small

reduce $\mathbf{y}$

Else put $\mathbf{y}$ into $\mathtt{B}_{\mathbf{x}}$

Sieve the bucket

$\forall$ pairs $\mathbf{y}, \mathbf{y}', \in \mathtt{B}_{\mathbf{x}}$:

If $\|\mathbf{y} \pm \mathbf{y}'\|$ – small

perform the reduction

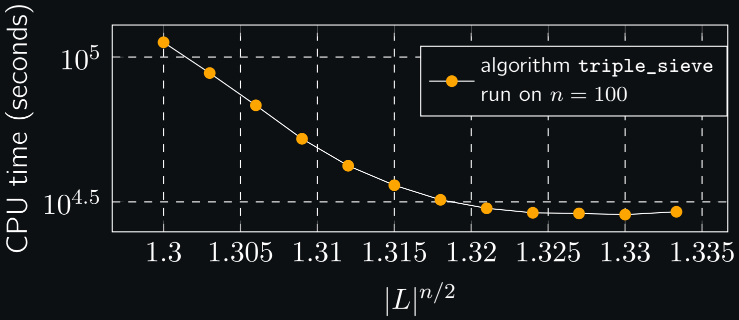
Else if $\|\mathbf{x} \pm \mathbf{y} \pm \mathbf{y}'\|$ – small

perform the reduction

- tuning the parameters allows to interpolate btw. 2-Sieve and 3-Sieve
- Smaller list $\Longrightarrow$ more 3-reductions
- Larger list $\Longrightarrow$ more 2-reductions

Time-memory trade-offs

With the same $|L|$, `triple_sieve` finds more reductions than 2-Sieve. It allows to decrease $|L|$.



The right most point corresponds to the 2-Sieve memory regime
The left most – to the 3-Sieve memory regime

Parallelized `triple_sieve`

- Again parallelization is done by having different threads work with different buckets.
- Reading the database of vectors is lock free
- Insertions of new shorter vectors into the global list are delayed and are executed in batches

Part IV

The G6K framework

The G6K framework

G6K includes previous and introduces new improvements for sieving:

1. Progressive sieving. [Duc18,ML18]: iteratively sieve in projected sublattices of smaller dimension
2. Dimensions for free, [Duc18]: sieve in a projected sublattice, Babai-lift short vectors
3. BKZ with sieving: recycle short vectors from one projected lattice to another (do not start from scratch!)

The G6K framework

G6K includes previous and introduces new improvements for sieving:

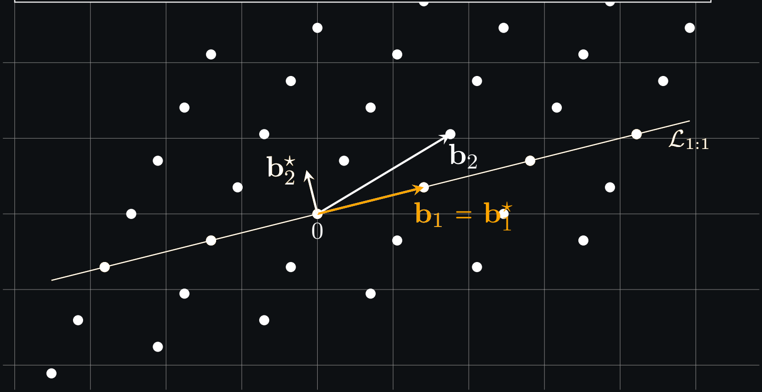
1. Progressive sieving. [Duc18,ML18]: iteratively sieve in projected sublattices of smaller dimension
2. Dimensions for free, [Duc18]: sieve in a projected sublattice, Babai-lift short vectors
3. BKZ with sieving: recycle short vectors from one projected lattice to another (do not start from scratch!)

Sources of improvements:

- Sieve outputs not only one short vector but many short vectors
- Sieving tries to improve the "quality" of a basis rather than just finding a shortest vector
- "Quality" - length of Gram-Schmidt vectors

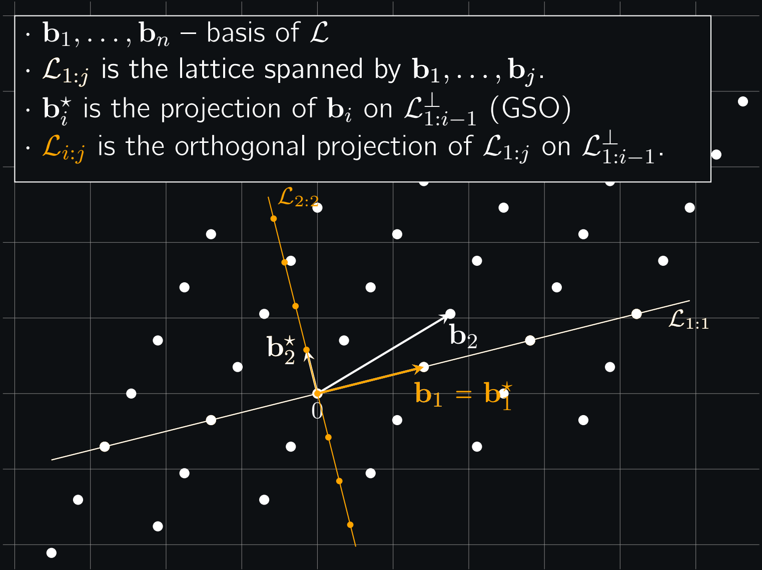
Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$
- $\mathcal{L}_{1:j}$ is the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_j$.



Projected lattice

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ – basis of $\mathcal{L}$
- $\mathcal{L}_{1:j}$ is the lattice spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_j$.
- $\mathbf{b}_i^\star$ is the projection of $\mathbf{b}_i$ on $\mathcal{L}_{1:i-1}^\perp$ (GSO)
- $\mathcal{L}_{i:j}$ is the orthogonal projection of $\mathcal{L}_{1:j}$ on $\mathcal{L}_{1:i-1}^\perp$.



Non black-box sieving in G6K

- Sieve in a projected sublattice $\mathcal{L}_{i:j}$ of rank $j - i + 1$. The output a list L of short vectors.
- Short vectors can be lifted from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i':j}$ for $i' < i$.
- Particularly short lifts are inserted into the current basis
- Move from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i':j'}$ using the set of instructions
  - Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$
  - Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$
  - Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

Non black-box sieving in G6K

With this set of instructions

- Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$
- Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$
- Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

G6K implements

1. Progressive sieving, [Duc18,ML18] iteratively sieve in $\mathcal{L}_{i:n}$ for decreasing i's.

Non black-box sieving in G6K

With this set of instructions

- Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$
- Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$
- Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

G6K implements

1. Progressive sieving, [Duc18,ML18] iteratively sieve in $\mathcal{L}_{i:n}$ for decreasing i's.
2. Dimensions for free, [Duc18]: sieve in $\mathcal{L}_{f:n}$ until enough short vectors are found, lift to $\mathcal{L}_{1:n} = \mathcal{L}$.
   For $f = \mathcal{O}(n/\lg n)$, lifts include the shortest vector.

Non black-box sieving in G6K

With this set of instructions

– Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$

– Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$

– Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

G6K implements

1. Progressive sieving, [Duc18,ML18] iteratively sieve in $\mathcal{L}_{i:n}$ for decreasing i's.
2. Dimensions for free, [Duc18]: sieve in $\mathcal{L}_{f:n}$ until enough short vectors are found, lift to $\mathcal{L}_{1:n} = \mathcal{L}$.
   For $f = \mathcal{O}(n/\lg n)$, lifts include the shortest vector.
3. Pumping: progressive sieve from $\mathcal{L}_{n:n}$ to $\mathcal{L}_{i:n}$, insert $n-i+1$ short vectors.

Non black-box sieving in G6K

With this set of instructions

– Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$

– Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$

– Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

G6K implements

1. Progressive sieving, [Duc18,ML18] iteratively sieve in $\mathcal{L}_{i:n}$ for decreasing i's.
2. Dimensions for free, [Duc18]: sieve in $\mathcal{L}_{f:n}$ until enough short vectors are found, lift to $\mathcal{L}_{1:n} = \mathcal{L}$.
   For $f = \mathcal{O}(n/\lg n)$, lifts include the shortest vector.
3. Pumping: progressive sieve from $\mathcal{L}_{n:n}$ to $\mathcal{L}_{i:n}$, insert $n-i+1$ short vectors.
4. Workout: execute Pump for decreasing i's.

Non black-box sieving in G6K

With this set of instructions

- Lifting: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i-1:j}$
- Inclusion: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i:j+1}$
- Projection: moves from $\mathcal{L}_{i:j}$ to $\mathcal{L}_{i+1:j}$

G6K implements

1. Progressive sieving, [Duc18,ML18] iteratively sieve in $\mathcal{L}_{i:n}$ for decreasing i's.
2. Dimensions for free, [Duc18]: sieve in $\mathcal{L}_{f:n}$ until enough short vectors are found, lift to $\mathcal{L}_{1:n} = \mathcal{L}$.
   For $f = \mathcal{O}(n/\lg n)$, lifts include the shortest vector.
3. Pumping: progressive sieve from $\mathcal{L}_{n:n}$ to $\mathcal{L}_{i:n}$, insert $n - i + 1$ short vectors.
4. Workout: execute Pump for decreasing i's.
5. BKZ

BKZ (simplified)

$\mathcal{L}_{[\ell\,;\,r]}$ - orthogonal projection of $\mathcal{L}_{1:r}$ on $\mathcal{L}_{1:\ell-1}^{\perp}$

Input: $B = (\mathbf{b}_i), \beta$

for $k = 2 \ldots n-1$ **do**

  $\mathbf{b} \leftarrow \mathrm{SVP}(\mathcal{L}_{[k\,:\,\min\{k+\beta-1,n\}]})$

end for

if $\mathbf{b}$ is "short enough" **then**

  Insert $\mathbf{b}$ into B

  Remove lin. dependencies

end if

$$\begin{pmatrix} | & | & | & & | & | & & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \ldots & \mathbf{b}_\beta & \mathbf{b}_{\beta+1} & \ldots & \mathbf{b}_n \\ | & | & | & & | & | & & | \end{pmatrix}$$

$$\underbrace{\mathbf{b}_1 \; \mathbf{b}_2 \; \mathbf{b}_3 \; \ldots \; \mathbf{b}_\beta}_{\text{SVP}}$$

BKZ with Sieving

$\mathcal{L}_{[\ell:r]}$ - orthogonal projection of $\mathcal{L}_{1:r}$ on $\mathcal{L}^{\perp}_{1:\ell-1}$

Input: $B = (\mathbf{b}_i), \beta$

for $k = 2 \ldots n-1$ **do**

Sieve($\mathcal{L}_{[k;k+1]}$)

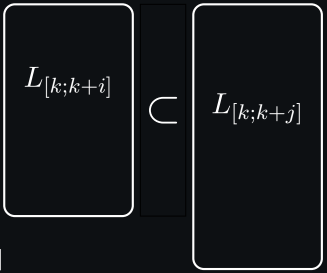
Sieve($\mathcal{L}_{[k;k+2]}$)

$\ldots$

Sieve($\mathcal{L}_{[k;k+\beta]}$)

end for

Update B with short $\mathbf{b}_i$'s from $L_{[k;k+\beta]}$



BKZ with Sieving

$\mathcal{L}_{[\ell:r]}$ - orthogonal projection of $\mathcal{L}_{1:r}$ on $\mathcal{L}^{\perp}_{1:\ell-1}$

Input: $B = (\mathbf{b}_i), \beta$

for $k = 2 \dots n-1$ **do**

Sieve($\mathcal{L}_{[k;k+1]}$)

Sieve($\mathcal{L}_{[k;k+2]}$)

$\dots$

Sieve($\mathcal{L}_{[k;k+\beta]}$)

end for

Update B with short $\mathbf{b}_i$'s from $L_{[k;k+\beta]}$

$k++$

$L_{[k;k+\beta]}$ short

$L_{[k;k+\beta]}$ short-ish

The last part

Experimental results

Experimental results (`bgj1_1`)

SVP dim	Hermite factor	Sieve max dim	Wall time	Total CPU time	Memory usage
155	1.00803	127	$14d\ 16h$	$1056d$	246 GiB
153	1.02102	123	$11d\ 15h$	$911d$	139 GiB
151	1.04411	124	$11d\ 19h$	$457.5d$	160 GiB
149	0.98506	117	$60h\ 7m$	$4.66kh$	59 GiB
147	1.03863	118	$123h\ 29m$	$4.79kh$	67.0 GiB
145	1.04267	114	$39h\ 3m$	$1496h$	37.7 GiB

On various machines with a lot of RAM (256 or 512 GiB).
The current record due to L. Ducas, M. Stevens, W. van Woerden: dim = 157

Implementation

The G6K is implemented as a C++ and Python library and is open-source.

`https://github.com/fplll/g6k`

The paper is available at

`https://eprint.iacr.org/2019/089`

Implementation

The G6K is implemented as a C++ and Python library and is open-source.

`https://github.com/fplll/g6k`

The paper is available at

`https://eprint.iacr.org/2019/089`

Thank you!

Q?

References

- [BDGL16] A. Becker, L. Ducas, N. Gama, T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving.
- [BGJ15] A. Becker,N. Gama, A. Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search.
- [Cha02] M. Charikar. Similarity estimation techniques from rounding algorithms.
- [Duc18] L. Ducas, Shortest vector from lattice sieving: A few dimensions for free.
- [FBB+15] R. Fitzpatrick, C.H.Bischof, J. Buchmann, O.Dagdelen, F.Göpfert, A. Mariano, and B. Yang, Tuning GaussSieve for speed.
- [HKL18] G. Herold, E. Kirshanova, T. Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving.
- [NV08] P. Nguyen, T. Vidick. Sieve algorithms for the shortest vector problem are practical.