Practical Improvements on BKZ Algorithm

Ziyu Zhao, Jintai Ding

Tsinghua University

Abstract. Lattice problems such as NTRU problem and LWE problem are widely used as the security base of post-quantum cryptosystems. And currently doing lattice reduction by BKZ algorithm is the most efficient way to solve them. In this paper, we give 4 further improvements on BKZ algorithm, which can be used for SVP subroutines base on enumeration and sieving. These improvements in combination provide a speed up of $2^{3\sim 4}$ in total. So all the lattice-based NIST PQC candidates lose $3 \sim 4$ bits of security in concrete attacks. Using these new techniques, we solved the 656 and 700 dimensional ideal lattice challenge in 380 and 1787 thread hours, respectively. The cost of the first one (also used an enumeration based SVP subroutine) is much less than the previous records (4600 thread hours). With these improvements enabled, we can still simulate the new BKZ algorithm easily. One can also use the simulator to find the blocksize strategy (and the corresponding cost) that makes Pot of the basis (defined in Section 4.2) decrease as fast as possible, which means the length of the first basis vector decrease the fastest if we accept the GSA assumption. It is useful for analyzing concrete attacks on lattice-based cryptography.

1 Introduction

Lattice based cryptography is nowadays an important part of post-quantum cryptography. The security of it is mainly based on some lattice problems, such as learning with error problem (LWE) [10,18,27] and NTRU problem [14]. These problems can be reduced to approximate shortest vector problem (ASVP) [22,21,5,17], i.e. to find a relatively short vector given a lattice basis. And currently lattice attacks are the most efficient way to solve such problems, thus it is important to know the concrete hardness of ASVP.

Currently there are two types of algorithm to find short vector in the lattice given a lattice basis. One is SVP algorithm like enumeration [26,15,6,32,33,7] and sieving [11,24,19,20,3], which can find almost the shortest vector in the lattice but the cost is at least exponential in the dimension of lattice. These SVP algorithms can only be applied to lattice with a small dimension, when there are restricted resources. Another type of algorithm, for instance LLL algorithm [16,23] and BKZ [32,29] algorithm can work on high dimensional lattice in a realistic time. LLL algorithm is extremely fast and often used as preprocessing, BKZ algorithm gives a bridge from shortest vector in small dimension to short vector with the same root Hermite factor in high dimension.

BKZ algorithm was first proposed by Schnorr in the 80's. It does enumeration on local blocks to find short vector then insert the new vector in the basis. Larger local blocksize gives shorter vector and cost more time. In 2011, Chen-Nguyen used some pruning technique in the enumeration step, it makes BKZ algorithm with a higher local blocksize practicable [4]. In 2016, Yuntao Wang et al. proposed their improved progressive BKZ [2], they get an optimized blocksize strategy based on their simulation of the total enumeration cost. It starts with a small blocksize, and increases it in a well organized manner, makes the algorithm significantly faster.

In this paper, we will give four further improvements on BKZ algorithm. We replace the insertion in BKZ by a processing on the local projected lattice. Then we can use a jumping technique to move more than one step each time in the BKZ tours. In the last several tours of BKZ, we only run the SVP subroutine on the first several indexes, which saves half of the time of these tours, then we give an end of the reduction by running a much heavier SVP subroutine.

We applied these techniques in lattice reduction with a SVP subroutine based on both enumeration and sieving. We implemented the new BKZ algorithm and tested it on ideal lattice challenges (the ideal lattice structure is never used), the running result shows we get a speed up with a factor $2^{3\sim4}$, which may be further improved since we did not used a well organized progressive BKZ (for the enumeration based BKZ, our blocksize are simply $80, 88, 96, \cdots$). Moreover our new BKZ algorithm is still easy to simulate (as BKZ 2.0) if we know the behavior of the SVP subroutine well.

Road Map. In Section 2, we present some basic facts about lattice and introduce the notations. Then in Section 3, we will recall the developments of BKZ algorithm in the history. Our further improvements on BKZ will be given in Section 4. In Section 5, we simulate the new BKZ algorithm and use it to give a further analysis for the techniques mentioned in the previous section. The information about the lattice challenge results is in Section 6.

2 Preliminaries

Lattice is discrete subgroup in $\mathbb{R}^m$. A lattice L always admits an integral basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n\}$ such that each vector $\mathbf{v}$ in L can be represented uniquely as a integral linear combination of $\mathbf{B}$, i.e. $\mathbf{v} = \lambda_1\mathbf{b}_1 + \cdots + \lambda_n\mathbf{b}_n$, $\lambda_i \in \mathbb{Z}$. We say n is the dimension of the lattice. The determinant of the lattice is defined to be $\sqrt{\det(\mathbf{B}\mathbf{B}^T)}$ which is equal to the absolute value of $\det(\mathbf{B})$ if $m = n$.

Gram-Schmidt Orthogonalization The Gram-Schmidt orthogonalization of $\mathbf{B}$ is given by $\mathbf{B}^* = (\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*)$ where $\mathbf{b}_i^*$ is defined by

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j^*,\ \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$$

we further denote by B_i the square of $\|\mathbf{b}_i^*\|$, we will call $[B_1, B_2, \cdots, B_n]$ the distance vector of the basis $\mathbf{B}$. The distance vector of a basis contains lots of information about it and this notation will be heavily used in the analysis of BKZ algorithm. We should also introduce the concept of local projected lattice. Let π_i be the orthogonal projection to $\mathrm{span}(\mathbf{b}_1, \cdots, \mathbf{b}_{i-1})^\perp$. Then we define the local projected lattice $L_{[i,j]}$ to be the lattice spanned by $B_{[i,j]} = (\pi_i(\mathbf{b}_i), \cdots, \pi_i(\mathbf{b}_j))$.

Gaussian Heuristic Because of the discreteness, the shortest nonzero vector in L exists (not unique in general). It is extremely hard to compute the shortest vector (proved to be NP-hard problem), but the length of it is estimated to be

$$\mathrm{GH}(L) = \frac{\Gamma(\frac{n}{2}+1)^{\frac{1}{n}}}{\sqrt{\pi}} \cdot \det(L)^{\frac{1}{n}} \approx \sqrt{\frac{n}{2\pi e}} \cdot \det(L)^{\frac{1}{n}}$$

when the lattice is "*random*" and n is not too small. In practice it works well if $n \geqslant 40$.

Root Hermite Factor For a vector $\mathbf{v}$ in a n dimensional lattice L, we define the root Hermite factor to be

$$\delta = \mathrm{rHF}(\mathbf{v}) = \left(\frac{\|\mathbf{v}\|}{\det(L)}\right)^{\frac{1}{n}}$$

as in [8], the root Hermite factor measures the quality of the vector. The hardness to get a vector of certain length mainly depends on its root Hermite factor.

HKZ reduced basis We say a lattice basis $\mathbf{B}$ of an n dimensional lattice L is HKZ-reduced if for each $1 \leqslant i \leqslant n$, the first vector of $B_{[i,n]}$ is the shortest nonzero vector of $L_{[i,n]}$.

3 History of BKZ Algorithm

3.1 The Original Algorithm

The first version of BKZ algorithm was proposed by Schnorr and Euchner as a generalization of the LLL algorithm [32]. Briefy, LLL algorithm gives the basis an order, then always reduces the latter vector by the former ones, and after the reduction done, it trys to make the former one shorter (in the corresponding local projected lattice) by swapping contiguous vector pairs. The algorithm terminates when no more swap or reduction can be done. The first vector of the output basis is of length about 1.02^n times the Gaussian heuristic in practice (see [8]), and the running time is polynomial in n, where n is the dimension of the lattice.

BKZ algorithm replaces the *swap* in LLL algorithm by a full enumeration in the local projected lattice to get shorter (also in the corresponding local projected lattice) vector. This vector will be inserted into the basis at a preselected place, and then we use an LLL algorithm to remove the linear dependency. The size of the local projected lattice is fixed and the place to do enumeration is pre-specified. Same as LLL algorithm, BKZ algorithm terminates when no nontrivial insertion can be done. The algorithm works as follows:

Algorithm 1: BKZ algorithm

```
   Input: a basis $B = (\mathbf{b}_1, \cdots, \mathbf{b}_n)$, blocksize $d$ and $\delta < 1$
   Output: A BKZ-$d$ reduced basis
1  LLL(B, $\delta$);
2  while last epoch did a nontrivial insertion do
3      for $i = 1, 2, \cdots, n-1$ do
4          $h = \max(i + d - 1,\ n)$;
5          $\mathbf{v} = \text{full\_enum}(L_{[i,h]})$; //find shortest nonzero vector by enumeration
6          if $\|\mathbf{v}\| < \delta\|\mathbf{b}_i\|$ then
7              LLL($\mathbf{b}_1, \cdots, \mathbf{b}_{i-1}, \mathbf{v}, \mathbf{b}_i, \cdots, \mathbf{b}_{\max(h+1,n)}, \delta$);
8              remove the zero vector in the first place.
9          else
10             LLL($\mathbf{b}_1, \cdots, \mathbf{b}_{\max(h+1,n)}, \delta$);
```

The running time of BKZ algorithm increase while the blocksize increase. It is not proved to be polynomial in n (the dimension) for fixed blocksize. But for small blocksize d (for example $d < 20$), the algorithm always terminates in reasonable time and the output quality is significantly improved. For $d = 20$ and n sufficiently large, the shortest nonzero vector it founds has length around 1.0128^n times the Gaussian heuristic (see [8]).

3.2 BKZ2.0

In 2011, Chen-Nguyen gave several improvements on the original BKZ algorithm [4], which made BKZ algorithm with a high blocksize ($d \sim 100$) practicable. The root Hermite factor of the output vector is improved to about 1.0095. They mainly did the following:

The first point is that, for a large blocksize d ($d \geqslant 40$), before no new insertion can be done, there is a long time that the quality of the basis improves poorly [12]. So they used the *early abort* technique to stop the algorithm as soon as the quality of the basis does not improve further. This provides an exponential speed up in practice [8] without degenerating the output quality.

They also did some modification on the enumeration step. For a larger blocksize d, experiments shows the running time of BKZ is dominanted by the enumeration subroutines. They used pruned enumeration instead of the full enumeration. A proper pruning can give an exponential speed up (about $2^{d/4}$) while the algorithm still output the shortest vector with a high probability. They further gave an *extreme pruning* technique [9], which repeat a further pruned enumeration output the shortest vector with a low probability for several times. This leads to a speed up of $2^{d/2}$.

Another thing they did is to preprocess the basis before doing the enumeration. Since the nodes to enumerate will be fewer if the basis has a better quality. Taking some time to do a light reduction will largely reduce the total enumeration time. In BKZ 2.0 they chose a BKZ algorithm with a small blocksize as preprocessing.

3.3 Progressive BKZ

Progressive BKZ mainly means to progressively enlarge the blocksize while doing reduction. The key idea is if an enumeration with low dimension can further reduce the lattice, there is no need to use a much larger dimension since the cost of SVP is at least exponential in the dimension. This technique was mentioned in several studies including [4,31,34,13]. These works mainly differ from the way they increase the blocksize. In 2016, [2] did a precise cost estimation of the progressive BKZ, and gave an optimized blocksize strategy. In their estimation, to do a BKZ-100 in an 800 dimensional lattice, their progressive BKZ is $2^{2.7}$ times faster than BKZ 2.0. And it's estimated to be 50 times faster than BKZ 2.0 for solving SVP challenges (see [28]) up to dimension 160.

4 Several Improvements about BKZ Algorithm

In this section we will give several techniques to further accelerate the BKZ algorithm. These techniques can be used for BKZ based on both sieving and enumeration SVP subroutines. And one can use it almost for free (except for large final run for sieving, which requires more memory) to get a considerable acceleration in practice.

4.1 Local Basis Processing Instead of Insertion

Currently we have two types of SVP algorithms, enumeration and sieving. Sieving is faster but requires large space which grows exponentially in the sieving dimension. To do a large sieving or enumeration once with the hope of finding the shortest vector is generally not the best choice.

The number of nodes on the enumeration tree grows as the basis gets worse [9]. In practice, it's better to do some preprocessing as in BKZ2.0. So the whole enumeration process not only gives a short vector, but also a rather good basis. For sieving, one often use the left progressive sieve to accelerate, whose speed also relies on the quality of the basis. And one needs the first several entries in the distance vector to be small to get a large dimension for free, which saves both time and memory (for sieving techniques, see [1]). Thus it will not lead to much further cost to get a good basis.

Only insert one short vector like the original BKZ algorithm or BKZ2.0 will waste the almost *free* basis. It's generally better to compute the transform matrix of local processing (on the local projected lattice) and apply it on the vectors of the original basis (succeed by a size reduction). Then the next local basis to apply the SVP algorithms is already only little bit worse than an HKZ-reduced basis. An obvious gain is we need no more preprocessing for it. Such *local basis processing* technique is folklore for sieving based BKZ, but it is necessary to point it out because it's the foundation of the next technique, and we first used it on enumeration based BKZ.

4.2 Jump By Two or More

After we do a local basis processing with blocksize d, the first vector will be short, and it's easy to see that the next few vectors is not too long also. If we make our working context jump to right by two indices or more (i.e. to work on $L_{[i+s,j+s]}$ after $L_{[i,j]}$), say we jump s steps after each SVP subroutine, we accelerate by factor s while not degenerating the quality much.

Here a crucial point is to use a blocksize slightly larger than we require. For instance, if we want to do a BKZ with blocksize d, we can choose a $d' = d + s - 1$, then every time we jump s steps. The result will not be worse than after a tour of BKZ-d without jump technique (actually better), if the output basis of the SVP subroutine has the similar quality as a *HKZ-reduced* basis. After the modification, the number of SVP subroutine is only $\frac{1}{s}$ as before, and for each subroutine, the cost is $2^{0.386(s-1)}$ (practically, when $d \sim 90$) as before if we use SVP algorithm based on 3-sieve. Take $s = 4$ we get a speed up of at least $2^{0.84}$.

To verify the effectiveness of this method, we generate a 400-dimensional random lattice with determinant 32768^{400}. Then we do a BKZ reduction to make the first basis vector has length $32768 \cdot 1.01046^{400}$, corresponding to blocksize 75. We use *Pump* in [1] as the SVP-subroutine (the expected dimension for free is set to be 12, so it behaves like HKZ-reduction), run one tour of BKZ with different maximal sieving dimension (MSD) and jumping steps. To measure the quality of a basis, we introduce the following notation:

$$\mathrm{Pot}(L) = \prod_{i=1}^{n} B_i^{n+1-i}$$

For a given lattice, Pot is an increase function in the root Hermite factor of the first vector in the basis, if we accept the GSA assumption [30]. So a better basis will have a smaller Pot. We present $\mathit{\Delta} \log_2 \mathrm{Pot}$ and the cost in Table 1 and 2.

Table 1. Jumping step = 1

MSD	68	69	70	71	72	73	74	75
cpu hours	2.30	2.74	3.27	3.88	4.69	5.69	6.93	8.52
$\mathit{\Delta} \log_2 \mathrm{Pot}$	463	758	1166	1400	1910	2254	2674	2949
$\frac{\mathit{\Delta} \log_2 \mathrm{Pot}}{\mathrm{cost}}$	201	277	357	361	407	396	385	346

Table 2. Different jumping steps

(MSD, jumping step)	(72, 1)	(73, 2)	(74, 3)	(75, 4)	(76, 5)	(77, 6)	(78, 7)
cpu hours	4.69	2.84	2.31	2.13	2.20	2.30	2.51
$\mathit{\Delta} \log_2 \mathrm{Pot}$	1910	1797	1787	1962	2059	2084	2241
$\frac{\mathit{\Delta} \log_2 \mathrm{Pot}}{\mathrm{cost}}$	407	633	773	920	930	906	858

From the tables we see simply a larger jumping step ($4 \sim 6$) will lead to a speed up of $2^{1.19}$. There is also a detailed analysis of jump based on simulation of BKZ algorithm in the next section. The analysis there shows we can generally get a speed up of $2^{1.65}$ if we use an HKZ-reduction with time complexity $2^{0.386d}$ as the SVP subroutine.

We notice that in [1] they also tried a technique called *PumpNJump*. They set the jumping step to be 3, and the gain is about $2^{0.2}$ (estimated form Fig. 4 in their paper) when the blocksize is high. We guess it's mainly because they did not use a large enough blocksize.

4.3 Reduce Only When We Need

In practice, we usually don't need the whole BKZ reduced basis. What we want is just a short vector (in lattice challenge) or make the tail of the distance vector large enough such that we can get the key hidden in the lattice by a size reduction (in real attack of lattice based cryptography). We only introduce the case for lattice challenge here, because the other case is similar.

For example, if we want to do BKZ-d on an n-dimensional lattice. For the last $\left[\frac{n}{d}\right]$ tours of the algorithm, we don't need to visit all indexes. In fact we work as following:

```
Algorithm 2: The last several tours of our BKZ
  Input: an $n$-dimensional lattice $L$, blocksize $d$ and an SVP algorithm
  Output: a reduced basis, denote by $L_2$
1 $m = \left[\frac{n}{d}\right]$;
2 for $k = 1, 2, \cdots, m$ do
3     //do a BKZ tour on $L_{[1,n-kd+1]}$
4     for $i = 1, 2, \cdots, n-kd+1$ do
5         reduce $L_{[i,i+d-1]}$ by the SVP algorithm;
6 return $L$
```

While the original BKZ works as following:

```
Algorithm 3: The original BKZ
  Input: an $n$-dimensional lattice $L$, blocksize $d$ and an SVP algorithm
  Output: a reduced basis, denote by $L_1$
1 $m = \left[\frac{n}{d}\right]$;
2 for $k = 1, 2, \cdots, m$ do
3     for $i = 1, 2, \cdots, n-d+1$ do
4         reduce $L_{[i,i+d-1]}$ by the SVP algorithm;
5 return $L$
```

Because the BKZ tour on $L_{[1,n-kd+1]}$ only relates to the first $n-(k-1)d$ vectors, the first $n - \left[\frac{n}{d}\right] d + 1$ vectors of L_1 and L_2 are the same. But we save at least half of the time for these final BKZ tours.

Notice that if we use a progressively larger blocksize, even if we jump by two or more usually we stay on one dimension for only $1 \sim 2$ tours. This technique at least saves constant ratio of time for the total challenge since currently the best SVP algorithm takes exponential time. We present the details of our 700 dimensional lattice challenge below. From the table we see this technique saves about 900 cpu hours for our 700 dimensional challenge, which costs 1787 cpu hours in total.

Table 3. Details for the end of our 700 dimensional challenge

MSD	working on	CPU time	a full tour time	min length
91	$L_{[1,578]}$	196.3h	206.4h	812896
92	$L_{[1,466]}$	201.5h	260.2h	805991
92	$L_{[1,354]}$	152.3h	258.9h	787811
93	$L_{[1,242]}$	146.0h	365.2h	755466
94	$L_{[1,130]}$	147.0h	651.9h	729162

4.4 A Large Final Run

In BKZ for high dimensional lattice, we can choose a much larger dimension d in the last SVP subroutine (working on $[1, d]$) to get a much shorter vector, to save the time for several tours of SVP subroutines with a normal blocksize. Since one tour costs $n/s \cdot T_{svp}$, which is much larger than an SVP subroutine, this method works well in practice (if we are searching for the secret key hidden in the lattice, just do a large enumeration or sieving at the tail of the basis). A detailed analysis based on simulation of BKZ is presented in next section. For our 700-dimensional lattice challenge, we ran an sieving based SVP algorithm on the first 124 vectors, and got a vector of length 659874.

5 Simulation and Further Analysis

It is easy to simulate the BKZ based on local processing if we know the behavior of the SVP algorithm. Actually the only thing we need is the (average) distance vector (denote by $[D_1, \cdots, D_d]$) of the basis of a lattice with det = 1 after running the SVP subroutine.

Algorithm 4: Simulation of BKZ algorithm

```
Input: a distance vector $[B_1, B_2, \cdots, B_n]$, blocksize $d$ and the average distance vector $[D_1, \cdots, D_d]$
Output: the new distance vector after run one tour of BKZ-$d$
1 for $s = 1, \cdots, n - d + 1$ do
2     det $= (\prod_{i=s}^{s+d-1} B_i)^{\frac{1}{d}}$;
3     for $i = 0, \cdots, d - 1$ do
4         $B_{i+s} = \det \cdot D_{i+1}$;
```

5.1 Jump By Two or More

We take a reduced 700 dimensional lattice (from ideal lattice challenge) and compute its distance vector, then compare the jumping strategies. The determinant is 1023.35^{700} and the root Hermite factor of the first vector is 1.010205, which corresponds to blocksize ≈ 80. We generate the *ideal* distance vector of the HKZ-reduced basis of dimension from 80 to 93 by Algorithm 5, which is based on Gaussian Heuristic (this distance vector will be slightly better than the average of real samples, since for real samples sometimes the algorithm unluckily failed to find the shortest one, but it can not find a vector shorter than the shortest vector even if we are very lucky. This will not affect the results since we are only interested in the speed up ratio).

We run the simulation for different dimensions and different jumping steps, the results are in the Table 4 and 5. We set the cost of an 80 dimensional subroutine to be 1, and the cost of $80 + i$ dimensional subroutine to be $2^{0.386i}$ (based on the performance of 3-sieve in practice, dim ~ 90).

Algorithm 5: generate *ideal* distance vector

```
Input: dimension $d$ and an average distance vector $[D_1, \cdots, D_{60}]$ for 60-dimensional HKZ-basis of lattices
       with det = 1 (get from real samples)
Output: the ideal distance vector of dimension $d$
1 det = 1.0;
2 for $i = 1, \cdots, d-60$ do
3     $A_i = \det^{\frac{1}{d-i+1}} \cdot \frac{\Gamma(\frac{d-i+1}{2}+1)^{\frac{1}{d-i+1}}}{\sqrt{\pi}}$;
4     det = det $/A_i$;
5 det = $\det^{\frac{1}{60}}$;
6 for $i = d-59, \cdots, d$ do
7     $A_i = D_{i-d+60} \cdot \det$;
8 return $[A_1, \cdots, A_d]$
```

Table 4. Jumping step = 1

blocksize	80	81	82	83	84	85	86	87	88	89	90	91	92	93
cost	622	811	1059	1381	1802	2351	3076	4002	5221	6811	8886	11594	15125	19733
$\Delta \log_2 \text{Pot}$	-408	179	788	1417	2069	2741	3434	4149	4884	5641	6418	7217	8036	8875
$\frac{\Delta \log_2 \text{Pot}}{\text{cost}}$	-0.66	0.22	0.74	1.03	1.15	1.17	1.12	1.04	0.94	0.83	0.72	0.62	0.53	0.45

Table 5. Best jumping step for different blocksize

blocksize	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94
optimal step	1	1	2	3	4	5	6	7	8	9	9	10	12	12	13
$\frac{\Delta \log_2 \text{Pot}}{\text{cost}}$	-0.66	0.22	0.91	1.72	2.43	3.00	3.40	3.62	3.68	3.62	3.47	3.26	3.00	2.73	2.45

Table 4 shows the cost and the change of $\log_2 \text{Pot}$ after one tour of local process if we only jump one index each time. The result shows it's optimal to choose blocksize to be 85 (always assume we have enough RAM for sieving!) and $\log_2 \text{Pot}$ decreases 1.17 per cost.

Table 5 shows the optimal jumping step with different blocksize. We can see if we take the blocksize to be a little bit larger, we can decrease the $\log_2 \text{Pot}$ 3.68 per cost. The jumping technique lead to a speed up of $3.68/1.17 \approx 2^{1.65}$.

5.2 A Large Final Run

If we accept the GSA assumption [30], Pot will be an increase function in the root Hermite factor of the first vector in the basis. So always choosing the blocksize and jumping step such that Pot of the lattice decrease the fastest is optimal in certain sense. For the same 700 dimensional lattice, we used a brute force search to find the following optimal reduction path (based on the ideal distance vector generated by *Algorithm 5*, and we still set the cost of d dimensional SVP to be $2^{0.386(d-80)}$):

Table 6. The optimal BKZ path

num tours	blocksize	step	cost	total cost	$\|\mathbf{b}_1\|$
1	88	8	663	663	1236685
2	88	7	757	1420	1224940
3	88	7	757	2177	1212026
4	89	8	867	3044	1194810
5	89	7	989	4032	1182845
6	89	7	989	5021	1170149
7	90	8	1133	6154	1153534
8	90	8	1133	7287	1141592
9	90	7	1292	8579	1130924
10	91	8	1480	10059	1114683
11	91	8	1480	11539	1102578

If we want a vector with norm 1100000 by BKZ tours, the table suggests the cost is more than 11539. But a straightforward computation based on Gaussian Heuristic shows that an SVP subroutine on the first 110 vector can also do this, which only costs 3061. So the large final run saves the time of the final 7 tours of BKZ in this example.

We can search the optimal time to do a final run by a brute force search since the cost of simulation is negligible. For instance if we want a vector of length 350000, we have Figure 1.

After each tour, we draw the current cost (blue points) and the total cost if we use a final run to get a vector shorter than 350000 now (red points) on the graph. It shows that if we do a final run when the length is about 381000 the total cost will be $2^{26.67}$, less than $2^{27.68}$ if we simply run the BKZ tours. We can save more than half of the time with this technique (the time to get BKZ-80 reduced is negligible here). Note that with a sieving based SVP algorithm, this technique will cost more RAM. In this example, the blocksize of the final run will be 140, and we need blocksize only 127 if we simply do BKZ tours, so the RAM we need grows $2^{2.7}$ times. Anyway, this is completely free for enumeration based algorithms.

6 The Lattice Challenges

The ideal lattice challenge [25] was started at 2012. It provides many different ideal lattices with dimension up to 1024. The original goal of this challenge is to test the algorithms for finding short

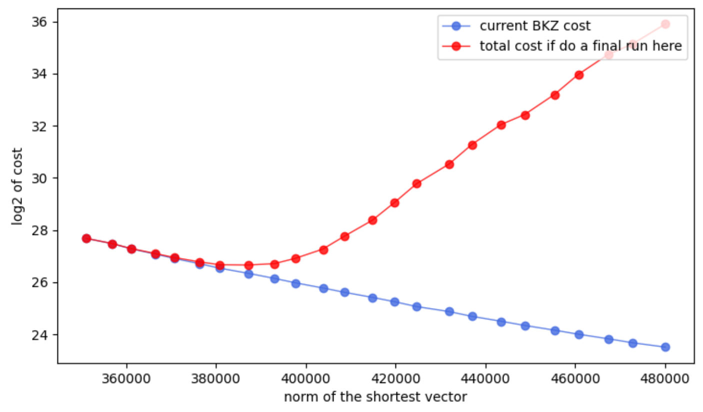

Fig. 1. Part of the optimal reduction path

vector in ideal lattice. But we will treat it as high dimensional random lattices to test our BKZ techniques. For each given lattice, a vector shorter than $1.05 \cdot \mathrm{GH}(L)$ can enter SVP Hall of Fame and a vector shorter than $n \cdot \det(L)^{\frac{1}{n}}$ can enter Approximate SVP Hall of Fame. We solved the following challenges:

Table 7. The ideal lattice challenges we solved

dimension	length	root Hermite factor	total cost	SVP subroutines based on
656	670275	1.00993	380 thread hours	Enumeration
700	659874	1.00928	1787 thread hours	Sieving

First we want to explain the reason to use ideal lattices. It is our duty to prove the effectiveness of our techniques by a large open lattice challenge. It is easy to cheat if one use a self-generated lattice or a toy example. And the ideal lattice challenge is the only proper one to test BKZ techniques, because the other three challenges require to do heavy reduction on a lattice with dimension $<$ 300, mainly depend on SVP algorithms. Moreover, early works on BKZ techniques such as [2] also used ideal lattice challenge. It's easy to compare if we use the same challenge. And as we know, no significant gain can be get from the ideal lattice structure in such a large dimension.

6.1 The Challenge Based on Enumeration

The 656-dimensional challenge was first finished in the summer of 2021, we used a laptop with Intel Core i7-7500U cpu (2.70 GHz) and a non-optimized c++ program which was modified several times while running. The total cost is about 700 core-hours (the information uploaded to the website of ideal lattice challenge is wrong). Later we optimized the program and ran it on a Xeon Silver 4208 CPU (2.10GHz) again. It takes only 380 core hours, much faster than the previous record which takes 4637 thread hours to solve a 652-dimensional approximate SVP challenge.

For the SVP subroutine, to get a reduced basis we used a variant of DeepBKZ [35]. DeepBKZ replaces the LLL in the original BKZ algorithm by DeepLLL (see [32]), which allows an operation called *deep insertion*. We modified the algorithm to further check all short vectors from an enumeration if it can be inserted into some former place, and always choose the candidate that can be inserted the deepest. For 20 random lattices (dim = 96) with determinant 1000, we run our modified

DeepBKZ for 800 seconds, and compute the average of the distance vectors, we take square root of each element and presented it below (the Gaussian Heuristic is 2442):

[2515 2519 2491 2442 2445 2409 2378 2352 2301 2271 2255 2208 2184 2152 2113 2080 2059 2041 1981 1948 1939 1882 1871 1842 1826 1790 1761 1735 1689 1670 1624 1585 1583 1538 1521 1467 1445 1401 1356 1344 1303 1269 1249 1213 1192 1153 1128 1110 1061 1044 1010 991 955 929 899 880 865 837 819 800 760 746 728 705 684 665 652 630 618 595 580 563 541 533 518 504 491 479 466 459 448 433 417 412 399 386 380 362 360 352 345 341 332 323 321 327]

These shows our SVP algorithm performs essentially on the same order of magnitude as [2]'s, the acceleration of the whole task ($4637/380 \approx 2^{3.6}$) mainly comes from the techniques in the previous sections. And we did not use a highly optimized blocksize strategy which may give a further speed up (the blocksize we used was simply 80, 88, 96, $\cdots$).

6.2 The Challenge Based on Sieving

The 700-dimensional challenge was finished recently. We start with a BKZ-80 reduced basis (take about 200 thread hours to get it), and then used a sieving based BKZ. The Sieving step takes 1587 thread hours, also on the Xeon Silver 4208 CPU (2.10GHz). The jumping step was set to be 6, slightly different from the analysis in Section 5. That's because we chose a larger blocksize and allowed deeper insertions to get more *dimension for free* [1], thus the output basis of the SVP subroutine is much worse than an HKZ-reduced basis.

For the maximal sieving dimension, before each tour of BKZ we calculate current Pot and compute the corresponding blocksize d by the GSA assumption. We then choose the maximal sieving dimension to be d. Such a choice performs well in practice. The expected dimension for free is set to be 18. so we work on a local projected lattice with dimension $d + 18$.

7 Conclusion

For the lattice-based NIST PQC candidates, most of our techniques are practicable and reduce $3 \sim 4$ bits of security in the concrete attacks. If we also consider the memory overhead, *a large final run* which gives about 1 bit speed up may not be a good choice since it use significantly more memory. In the same time, in this paper we used $2^{0.386d}$ as the sieving cost (based on the experiments when the sieving dimension is about 90). For large sieving dimension, this cost should be $2^{0.292d}$ if we use the state-of-art sieving algorithm [3]. So for NIST candidates the speedup ratio of the jumping technique will be more than $2^{1.65}$ (for our low dimensional case). A precise estimation of the impact of these techniques on the NIST candidates may be a direction of the future work.

References

1. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: IACR Cryptol. ePrint Arch. (2019) 4.1, 4.2, 4.2, 6.2
2. Aono, Y., Wang, Y., Hayashi, T., Takagi, T.: Improved progressive bkz algorithms and their precise cost estimation by sharp simulator. In: EUROCRYPT (2016) 1, 3.3, 6, 6.1
3. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: IACR Cryptol. ePrint Arch. (2015) 1, 7
4. Chen, Y., Nguyen, P.Q.: Bkz 2.0: Better lattice security estimates. In: ASIACRYPT (2011) 1, 3.2, 3.3
5. Coppersmith, D., Shamir, A.: Lattice attacks on ntru. In: Fumy, W. (ed.) Advances in Cryptology — EUROCRYPT '97. pp. 52–61. Springer Berlin Heidelberg, Berlin, Heidelberg (1997) 1
6. Fincke, U., Pohst, M.E.: Improved methods for calculating vectors of short length in a lattice. Mathematics of Computation (1985) 1
7. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: Advances in Cryptology – Proceedings of EUROCRYPT '10. LNCS, vol. 6110. Springer (2010) 1

8. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: EUROCRYPT (2008) 2, 3.1, 10, 3.2
9. Gama, N., Nguyen, P.Q., Regev, O.: Lattice enumeration using extreme pruning. In: EUROCRYPT (2010) 3.2, 4.1
10. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432 (2007), https://ia.cr/2007/432 1
11. Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) Coding and Cryptology. pp. 159–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 1
12. Hanrot, G., Pujol, X., Stehlé, D.: Analyzing blockwise lattice algorithms using dynamical systems. In: CRYPTO (2011) 3.2
13. Haque, M.M., Rahman, M.O.: Analyzing progressive-bkz lattice reduction algorithm. International Journal of Computer Network and Information Security (2019) 3.3
14. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. pp. 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1998) 1
15. Kannan, R.: Improved algorithms for integer programming and related lattice problems. Proceedings of the fifteenth annual ACM symposium on Theory of computing (1983) 1
16. Lenstra, A.K., Lenstra, H.W., Lovász, L.M.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 515–534 (1982) 1
17. Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: Kiayias, A. (ed.) Topics in Cryptology – CT-RSA 2011. pp. 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 1
18. Micciancio, D., Regev, O.: Lattice-based Cryptography, pp. 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-540-88702-7_5 1
19. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem 1
20. Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. Electron. Colloquium Comput. Complex. 17, 14 (2010) 1
21. Nguyen, P.Q.: Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97. In: CRYPTO (1999) 1
22. Nguyen, P.Q., Regev, O.: Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. In: Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 4004, pp. 271–288. Springer (2006), https://iacr.org/archive/eurocrypt2006/40040273/40040273.pdf 1
23. Nguyen, P.Q., Valle, B.: The lll algorithm - survey and applications. In: Information Security and Cryptography (2010) 1
24. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. Journal of Mathematical Cryptology 2(2), 181–207 (2008), https://doi.org/10.1515/JMC.2008.009 1
25. Plantard, T., Schneider, M.: Creating a challenge for ideal lattices. Cryptology ePrint Archive, Report 2013/039 (2013), https://ia.cr/2013/039 6
26. Pohst, M.E.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. SIGSAM Bull. 15, 37–44 (1981) 1
27. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: In STOC. pp. 84–93. ACM Press (2005) 1
28. Schneider, M., Gama, N.: Darmstadt svp challenges (2010) 3.3
29. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. 53, 201–224 (1987) 1
30. Schnorr, C.P.: Lattice reduction by random sampling and birthday methods. In: STACS (2003) 4.2, 5.2
31. Schnorr, C.P.: Accelerated and improved slide-and lll-reduction (2012) 3.3
32. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming 66, 181–199 (1994) 1, 3.1, 6.1
33. Schnorr, C.P., Hörner, H.H.: Attacking the chor-rivest cryptosystem by improved lattice reduction. In: Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding. Lecture Notes in Computer Science, vol. 921, pp. 1–12. Springer (1995) 1
34. Schnorr, C.P., Shevchenko, T.: Solving subset sum problems of densioty close to 1 by "randomized" bkz-reduction. IACR Cryptol. ePrint Arch. 2012, 620 (2012) 3.3
35. Yamaguchi, J., Yasuda, M.: Explicit formula for gram-schmidt vectors in lll with deep insertions and its applications. In: NuTMiC (2017) 6.1