SOLUTIONS OF HARD KNAPSACK PROBLEMS USING EXTREME PRUNING

E. DARAVIGKAS[1], K.A. DRAZIOTIS[2], AND A. PAPADOPOULOU[3]

Abstract. In the present study we provide a review for the state of the art attacks to the knapsack problem. We implemented the Schnorr-Shevchenko lattice attack and we applied the new reduction strategy, BKZ 2.0. Finally, we compared the two implementations.

1. Introduction

In this work we address the class of knapsack problems with density close to 1. This problem except its cryptographic applications [21], is also very interesting in computational number theory. The last years there was a better understanding of lattice reductions and new methods were discovered [3]. This, had a dramatic effect in the cryptanalysis of lattice based cryptosystems, changing the security parameters to some of them [26]. We shall apply these new implementations to a lattice attack presented in [25]. As far as we know, these are the first experiments in this direction.

We start by describing the problem. Given a list of n positive integers $\{a_1, \ldots, a_n\}$ and an integer s such that

$$\max\{a_i\}_i \leq s \leq \sum_{i=1}^{n} a_i,$$

find a binary vector

$$\mathbf{x} = (x_i)_i \text{ with, } \sum_{i=1}^{n} x_i a_i = s. \tag{1.1}$$

We define the density of the knapsack (or subset sum problem) to be,

$$d = \frac{n}{\log_2 \max_i\{a_i\}_i}.$$

The decision version of the problem is known to be NP-complete [12]. In cryptography we are interested in values of d less than one. Since if $d > 1$, then there are

2010 *Mathematics Subject Classification.* 11Y16, 94A60, 11D04.

Key words and phrases. knapsack problem, Subset Sum problem, Lattice, LLL reduction, BKZ reduction, extreme pruning.

[1] Aristotle University of Thessaloniki, Department of informatics, vaggdar@gmail.com.

[2] Aristotle University of Thessaloniki, Department of informatics, drazioti@csd.auth.gr.

[3] Aristotle University of Thessaloniki, Department of Mathematics, anastasia3g@hotmail.com.

Appeared in Modern Discrete Mathematics and Analysis pp. 81-95. Springer Optimization and Its Applications book series (SOIA, volume 131, 2018)

https://doi.org/10.1007/978-3-319-74325-7_4

many solutions of the knapsack problem and so we can not transmit efficient information. Also, because of the low density attacks [2, 15], we consider the density close to 1.

Roadmap. In section 2 we provide some basic facts about lattices and the LLL-reduction. In the next section we describe the possible attacks to the knapsack problem and we present the BKZ-reduction which is a generalization of LLL-reduction. Finally, in section 3 we present our experiments. In section 4 we provide some concluding remarks.

2. Lattices

For an introduction to lattices see [7, 14, 18].

Definition 1. A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^m$.

So a lattice is a subgroup of the Abelian group $\langle \mathbb{R}^m, + \rangle$ thus $\mathbf{0} \in \mathcal{L}$, and is discrete as a subset of $\mathbb{R}^m$. That is $\mathbf{0}$ is not a limit point. This means that the intersection of $\mathcal{L}$ and any bounded set of $\mathbb{R}^m$ is a finite set of points. For instance $\{\mathbf{0}\}$ and every subgroup of the additive group $\langle \mathbb{Z}^m, + \rangle$ are lattices of $\mathbb{R}^m$. A set of vectors $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^m$ (where $n \leq m$), is called generator of $\mathcal{L}$ if each vector of $\mathcal{L}$, is a linear combination of vectors in $\mathcal{B}$ with integer coefficients. If the vectors of this set are also independent, we call $\mathcal{B}$ a basis of the lattice. In this case the number n is the same for all bases and we call it *rank* of the lattice, i.e. $n = \dim(span(\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}))$. The integer m is called *dimension* of the lattice. If $n = m$, then we call the lattice, *full rank lattice*. Some authors call the rank as the dimension of the lattice. Usually we consider lattices which contain vectors only with integer coordinates. Such lattices are called *integer lattices*. A basis $\mathcal{B}$ is given by an $n \times m$ matrix, with rows the vectors of a basis of a lattice. If we have two distinct basis-matrices B and B', then there is a unimodular integer (square) matrix U with dimension n, such that $B = UB'$. It turns out that the Gram determinant $\det(BB^T)$ does not depend on the basis B or B'. This number is always positive and its square root is called determinant or volume of the lattice $\mathcal{L}$.

Definition 2. We call determinant or volume of the lattice $\mathcal{L}$ the positive real number

$$\det(\mathcal{L}) = vol(\mathcal{L}) = \sqrt{\det(BB^T)}.$$

The volume is in fact the (geometric) volume of the parallelepiped spanned by any basis of the lattice.

If $\mathcal{L}$ is a full rank lattice, then $vol(\mathcal{L}) = \det(B)$, where B is the square matrix representing a basis of $\mathcal{L}$. Having a matrix B we can write the lattice generated by the basis $\mathcal{B}$ as:

$$\mathcal{L}(\mathcal{B}) = \Big\{ \sum_{j=1}^{n} k_j \mathbf{b}_j : k_j \in \mathbb{Z} \Big\} \text{ or } \{\mathbf{x}B : \mathbf{x} \in \mathbb{Z}^n\}.$$

Since a lattice is a discrete subset of $\mathbb{R}^m$, then there is a nonzero vector $\mathbf{x} \in \mathcal{L}$ with the lowest Euclidean norm. We denote this norm by $\lambda_1(\mathcal{L})$ and we call it *first successive minima*. In general, not only one vector $\mathbf{v} \in \mathcal{L} - \{\mathbf{0}\}$ has $||\mathbf{v}|| = \lambda_1$. The number $|\{\mathbf{v} \in \mathcal{L} - \{\mathbf{0}\} : ||\mathbf{v}|| = \lambda_1\}|$ is called *kissing number* of the lattice. This number is at least 2 since, also $-\mathbf{v}$ has length λ_1. Further, Minkowski proved that $\lambda_1(L) \leq \sqrt{n} vol(L)^{1/n}$.

We set $\overline{B}_m(r) = \{\mathbf{x} \in \mathbb{R}^m : ||\mathbf{x}|| \leq r\}$. Then, $\lambda_1(\mathcal{L})$ is the smallest real number r such that, the real vector space $M_r(\mathcal{L}) = span(\mathcal{L} \cap \overline{B}_m(r))$, has at least one independent vector. Equivalently $\dim(M_r(\mathcal{L})) \geq 1$. In general we define $\lambda_j(\mathcal{L})$ to be the positive number $\inf\{r : \dim(M_r(\mathcal{L})) \geq j\}$. Always we have

$$\lambda_1(\mathcal{L}) \leq \lambda_2(\mathcal{L}) \leq \cdots \leq \lambda_n(\mathcal{L}).$$

An upper bound for the λ_i's was given by Minkowski.

Proposition 2.1. *(Second Theorem of Minkowski) For every integer $n > 1$ there exist a positive constant γ_n such that for any lattice with rank n and every $j \in \{1, 2, .., n\}$, we have the following inequality*

$$\Big(\prod_{i=1}^{n} \lambda_i\Big)^{1/n} \leq \sqrt{\gamma_n} vol(\mathcal{L})^{1/n}.$$

The problem of finding a shortest vector in a lattice is called Shortest Vector Problem (SVP) and is one of the most important and difficult problems in lattices. Since this problem is hard (is proved that is NP-hard under randomized reductions by Ajtai [1]), we try to attack the approximation problem. For instance we are looking for lattice vectors $\mathbf{v}$ such that $||\mathbf{v}|| < \gamma(n)\lambda_1(\mathcal{L})$, where $n = rank(\mathcal{L})$. In fact LLL, as we shall see in the next section, solves SVP for approximating factors $\gamma(n) = 2^{(n-1)/2}$.

Since lattice bases always exist, we are interested in bases with short and nearly orthogonal vectors. The procedure of finding such a basis (given one) is called lattice reduction.

2.1. **Lenstra-Lenstra-Lovász algorithm (LLL).** Before we discuss LLL algorithm which is a reduction algorithm for lattices in $\mathbb{R}^m$, we shall remind the reader the Gram-Schmidt procedure, which is a basic sub-routine in the LLL algorithm.

2.1.1. *Gram-Schimdt Orthogonalization (GSO).*

Definition 3. Let $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be independent set of vectors in $\mathbb{R}^m$. Then, their GSO is a set of orthogonal vectors $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*\}$ defined by the following relations

$$\mathbf{b}_1^* = \mathbf{b}_1,\ \mathbf{b}_i^* = proj_{M_{i-1}}(\mathbf{b}_i),\ 2 \leq i \leq n,$$

where $M_{i-1} = span(\{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}\})^{\perp}$, is the orthogonal span of $\{\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}\}$.

We rewrite the previous relations as

$$\mathbf{b}_1 = \mathbf{b}_1^*,\ \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*,\ \mu_{i,j} = \frac{\mathbf{b}_i \cdot \mathbf{b}_j^*}{||\mathbf{b}_j^*||^2},\quad 2 \leq j < i \leq n.$$

If we consider the orthonormal basis of $span(\mathcal{B})$,

$$\mathcal{B}' = \Big\{\mathbf{b}_1' = \frac{\mathbf{b}_1^*}{||\mathbf{b}_1^*||}, \mathbf{b}_2' = \frac{\mathbf{b}_2^*}{||\mathbf{b}_2^*||}, \ldots, \mathbf{b}_n' = \frac{\mathbf{b}_n^*}{||\mathbf{b}_n^*||}\Big\},$$

then the transition matrix $T = T_{\mathcal{B}' \leftarrow \mathcal{B}}$ is the upper triangular matrix $(n \times n)$

$$T = \begin{pmatrix} ||\mathbf{b}_1^*|| & \mu_{2,1}||\mathbf{b}_1^*|| & \cdots & \mu_{n,1}||\mathbf{b}_1^*|| \\ 0 & ||\mathbf{b}_2^*|| & \cdots & \mu_{n,2}||\mathbf{b}_2^*|| \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & ||\mathbf{b}_n^*|| \end{pmatrix} \tag{2.1}$$

and $(\mathbf{b}_1, \ldots, \mathbf{b}_n) = (\mathbf{b}'_1, \ldots, \mathbf{b}'_n)T$. If we consider the $n \times n$ lower-triangular matrix

$$\mu = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \mu_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{n,1} & \mu_{n,2} & \cdots & 1 \end{pmatrix}$$

then $(\mathbf{b}_1, \ldots, \mathbf{b}_n)^T = \mu(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)^T$. So the GSO of $\mathcal{B}$ is given by the vectors $(||\mathbf{b}_i^*||)_i$ and the triangular matrix μ.

GSO Pseudocode :
INPUT: A basis $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ of $V = span(\mathcal{L})$
OUTPUT: An orthogonal basis of V and the Gram-Schimdt matrix μ

01. $\mathbf{R}_1 \leftarrow \mathbf{b}_1$
02. **for** $i = 1$ to n
03. $\quad \mathbf{t}_1 \leftarrow \mathbf{b}_i$
04. $\quad \mathbf{t}_3 \leftarrow \mathbf{t}_1$
05. $\quad$ **for** $j = i - 1$ to 1
06. $\quad\quad \mu_{i,j} \leftarrow \frac{\mathbf{t}_1 \cdot \mathbf{R}_j}{||\mathbf{R}_j||^2}$
07. $\quad\quad \mathbf{t}_2 \leftarrow \mathbf{t}_3 - proj_{\mathbf{R}_j}(\mathbf{t}_1)$
08. $\quad\quad \mathbf{t}_3 \leftarrow \mathbf{t}_2$
09. $\quad$ **end for**
10. $\quad \mathbf{R}_j \leftarrow \mathbf{t}_2$
11. **end for**
12. Return $R = (\mathbf{R}_1, \ldots, \mathbf{R}_n)$, $\mu = ([\mu_{i,j}])_{1 \le i,j \le n}$

The running time of GSO is polynomial with respect to $\max\{n, \log \max_i ||\mathbf{b}_i||\}$.

2.1.2. *LLL-pseudocode.*

Definition 4. A lattice basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is called *size-reduced* if and only if

$$|\mu_{i,j}| = \frac{|\mathbf{b}_i \cdot \mathbf{b}_j^*|}{||\mathbf{b}_j^*||^2} \le \frac{1}{2}, \ 1 \le j < i \le n.$$

$\mathcal{B}$ is called $\delta-$LLL reduced if in advance

$$\text{(Lovász condition) } \delta||\mathbf{b}_{i-1}^*||^2 \le ||\mu_{i,i-1}\mathbf{b}_{i-1}^* + \mathbf{b}_i^*||^2, \ 2 \le i \le n.$$

In LLL algorithm we first apply size reduction. That is, we consider vectors of the lattice near the Gram-Schimdt basis. This property introduced by Lagrange. Then, we continue by checking if Lovász condition does not hold, in two successive vectors, and in this case we swap the vectors.

LLL Pseudocode
INPUT: A basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{Z}^m$
of the lattice $\mathcal{L}(\mathcal{B})$ and a real number $\delta \in (1/4, 1)$
We use the function $gso = GSO[\mathcal{B}]$ which returns the orthogonal basis gso[0]
and the Gram-Schimdt matrix gso[1]

OUTPUT: δ-LLL-reduced basis for $\mathcal{L}$

```
--   Initialization:
01. i = 2
02. gso ← GSO(B)
03. (b_i)_i ← gso[1]
04. (μ_{i,j})_{i,j} ← gso[2]
--   Size Reduction Step:
05. While i ≤ n do
06.     for j = i − 1 to 1 do
07.         c_{i,j} ← ⌊μ_{i,j}⌉          #⌊x⌉ = ⌊x + 0.5⌋
08.         b_i ← b_i − c_{i,j} b_j
09.         update B and gso ← GSO(B)
10.         (b_i)_i ← gso[1]
11.         (μ_{i,j})_{i,j} ← gso[2]
12.     end for
--    Swap step:
13.     if δ||b*_i||^2 > ||μ_{i+1,i} b*_i + b*_{i+1}||^2 then
14.         b_i ↔ b_{i+1}
15.         i = max(2, i − 1)
16.         update B and gso ← GSO(B)
17.         (b_i)_i ← gso[1]
18.         (μ_{i,j})_{i,j} ← gso[2]
19.     else
20.         i ← i + 1
21.     end if
22.    end while
23. Return B
```

There are two basic efficient (and stable) implementations, using floating point arithmetic of suitable precision: NTL [26] and fpLLL [5]. A version of LLL using floating point arithmetic was first presented by Schnorr in [24] and later by Nguyen and Stehlé [19].

2.1.3. *Analysis of LLL.* LLL-reduction algorithm applies two basic operations to the initial basis. Size reduction and swap of two vectors, if Lovász condition fails. After the first operation, lines 06-12, we get $|\mu_{i,j}| < 1/2$. Then we check the Lovász condition, lines 13-20. According if the previous check is true or false we swap or not the vectors. This, schematically can be explained in terms of matrices. We consider the 2×2 block of matrix 2.1, indexed at position (i, i). That is

$$\begin{pmatrix} ||\mathbf{b}_i^*|| & \mu_{i+1,i}||\mathbf{b}_i^*|| \\ 0 & ||\mathbf{b}_{i+1}^*|| \end{pmatrix}.$$

Then Lovász condition holds, if the second column has length at least $\sqrt{\delta}$ times the length of the first column. The following theorem was proved in [16].

Theorem 2.2. *Let $\mathcal{L}(\mathcal{B})$ with $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, $\mathbf{b}_i \in \mathbb{Z}^m$, $\Lambda = \max_i(||\mathbf{b}_i||^2)$ and $\delta = 3/4$. Then the previous algorithm terminates after $O(n^4 \log \Lambda)$ arithmetic*

operations and the integers on which these operations are performed have bit length $O(n \log \Lambda)$.

Proof. Proposition 1.26 in [16]. □

Although the first proof about the complexity of LLL, concerns only integer lattices and $\delta = 3/4$ later was proved for every $\delta \in (1/4, 1)$. A basic consequence is the following proposition.

Proposition 2.3. *If* $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ *is a* $\delta-$*LLL reduced basis (with* $\delta = 3/4$*) of the lattice* $\mathcal{L}(\mathcal{B})$ *with* $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, $\mathbf{b}_i \in \mathbb{R}^m$. *Then,*

$$||\mathbf{v}_1|| \leq 2^{(n-1)/2} \lambda_1(\mathcal{L}).$$

Proof. [7, Lemma 17.2.12] □

So, LLL provides a solution to the approximate SVP_γ, with factor $\gamma = 2^{(n-1)/2}$.

2.2. **Blockwise Korkine-Zolotarev (BKZ).** In LLL algorithm we check Lovász condition in a matrix of blocksize 2. In BKZ algorithm Lovász condition is generalized to larger blocksizes. This has the positive effect of providing better quality of the output basis than in LLL, but with a cost in the running time. BKZ does not run in polynomial time, in fact Gama and Nguyen [10], experimentally provide strong evidences that BKZ runs exponentially with the rank of the lattice. The running time of the reduction increases significantly with higher blocksizes. However, increasing blocksizes also means an improvement in the quality of the output basis, i.e. basis with smaller length. If the blocksize is equal to the rank of the lattice, then we get a Hermite-Korkine-Zolotarev (HKZ) reduced basis.

BKZ is an approximation algorithm, which takes as input the blocksize β and a parameter δ (the same as in LLL) and then it calls an enumeration subroutine many times looking for a shortest vector in projected lattices of dimension $\leq \beta$. The cost of the enumeration subroutine is $2^{O(\beta^2)}$ operations [3].

We provide the description of BKZ and then its pseudocode. As we mentioned above, the input basis provides some local blocks with dimension β. The first block consists of the first basis vector and the $\beta - 1$ succeeding vectors. For the second block, we take the projections of the second basis vector and the succeeding vectors onto the orthogonal complement of the vector. The remaining blocks are constructed in the same way. The index l of the last vector in a block is computed as $l = \min(i + \beta - 1, n)$, where n is the rank of the lattice. We start with two definitions.

Definition 5. Let $\mathcal{L} = \mathcal{L}(\mathcal{B})$, be a lattice with basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. We define,

$$\pi_i : \mathcal{L} \to M_{i-1} = span(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$$

$$\pi_i(\mathbf{x}) = proj_{M_{i-1}}(\mathbf{x}).$$

With $\mathcal{L}_{[i,k]}$ we denote the lattice generated by the vectors

$$\mathcal{B}_{[i,k]} = \{\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), \ldots, \pi_i(\mathbf{b}_k)\},$$

for $1 \leq i \leq k \leq n$. Remark that $\pi_i(\mathcal{L}) = \mathcal{L}_{[i,n]}$ and $rank(\mathcal{L}_{[i,k]}) = k + 1 - i$.

For the given basis $\mathcal{B}$ we get $\pi_i(\mathbf{b}_j) = \mathbf{b}_j - \sum_{k=1}^{i-1} \mathbf{b}_k^*$.

Definition 6. (BKZ). We say that the basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ is BKZ$-\beta$ reduced with blocksize β and parameter δ, if it is $\delta-$LLL reduced ($\delta \in (1/4, 1)$) and for each $i \in \{1, 2, \ldots, n\}$ we have $||\mathbf{b}_i^*|| = \lambda_1(\mathcal{L}_{[i,k]})$ for $k = \min(i + \beta - 1, n)$.

The actual improvement is achieved as follows: In every step of the algorithm we ensure that the first vector of each block $\mathcal{B}_{[i,k]}$ is the shortest vector inside each lattice. If it is not, then the shortest vector of the lattice is inserted into the block, but this vector is linear dependent on the basis vectors. That means that the local block is not a basis for the local lattice any longer. For this reason, we use the LLL algorithm on the expanded block. The blocksize determines the dimension of most of the local projected lattices in which shortest vectors have to be found. It also determines the amount of basis vectors that are used as input for the LLL algorithm after insertion of a new vector into the local block.

The LLL algorithm reduces iteratively each local block $B_{[j,\min(j+\beta-1,n)]}$ for $j = 1, \ldots, n$, to ensure that the first vector is the shortest one. If $\beta = 2$, then BKZ coincides with LLL, since $B_{[j,\min(j+\beta-1,n)]} = B_{[j,j+1]}$. BKZ algorithm uses an enumeration algorithm to find a vector $\mathbf{v} = (v_1, \ldots, v_n)$ such that,

$$||\pi_j(\sum_{i=j}^{k} v_i \mathbf{b}_i)|| = \lambda_1(L_{[j,k]}).$$

In each iteration we check if $||\mathbf{b}_j^*|| > \lambda_1(L_{[j,k]})$. If it is true, then a new vector $\mathbf{b}' = \sum_{i=j}^{k} v_i \mathbf{b}_i$ is inserted in the basis and the vectors are linear dependent from now. Then, LLL is called to solve this problem and provide a reduced basis. The procedure ends when all of the enumerations fail.

The enumeration subroutine.

Enumeration constitutes a basic part of BKZ algorithm. It finds a shortest vector in a local projected lattice $L_{[j,k]}$. That means given as input two integers j, k such that $0 \leq j \leq k \leq n$ provides as output a vector $\mathbf{v} = (v_j, \ldots, v_k)$ such that $||\pi_j(\sum_{i=j}^{k} v_i \mathbf{b}_i)|| = \lambda_1(L_{[j,k]})$. The subroutine enumerates the vectors that are within a cycle, with radius $R = ||\mathbf{b}_j^*||$: an initial bound of $\lambda_1(L_{[j,k]})$.

In order to find a shortest vector, enumeration algorithm goes through the *enumeration tree* which consists of the half vectors in the projected lattices $L_{[k,k]}$, $L_{[k-1,k]}$, $\ldots$, $L_{[j,k]}$ of norm at most R. The reason why this tree formed only by the half number of vectors is the following: if $\mathbf{v} \in \mathcal{L}$, then $-\mathbf{v} \in \mathcal{L}$. Thus, we count only the *positive* nodes. The more reduced the basis is, the less nodes in the tree, and the enumeration is faster.

The tree has depth $k - j + 1$ and the root of the tree is the zero vector, while the leaves are all the vectors of L with norm $\leq R$. The parent of a node at depth k is at depth $k - 1$. Child nodes are ordered by increasing Euclidean norm. The Schnorr-Euchner algorithm performs a Depth First Search of the tree to find a non-zero leaf of minimal norm.

The process of enumeration is quite expensive (especially for large β) and for this reason we attempt not to enumerate all the tree nodes and discard some nodes. Pruning can speed-up the subroutine, but the output vector may not be a shortest one. Schnorr and Euchner first studied pruning enumeration and the basic idea is replacing the radius R, with $R_k < R$ for $k = 1, \ldots, n$.

The following algorithm is the Block Korkin-Zolotarev (BKZ), as proposed by Chen and Nguyen [3].

BKZ Pseudocode (without pruning):

INPUT: A basis $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of $\mathcal{L}$,
blocksize β, the Gram-Schmidt triangular matrix μ and
the lengths $||\mathbf{b}_i^*||^2, \ i = 1, \ldots, n$
OUTPUT: A BKZ-β reduced basis of $\mathcal{L}$

01. $z \leftarrow 0$
02. $j \leftarrow 0$
03. $LLL(\mathbf{b}_1, \ldots, \mathbf{b}_n, \mu)$
04. **while** $z < n$
05. $\quad j \leftarrow (j \pmod{n-1} + 1)$
06. $\quad k \leftarrow \min(j + \beta - 1, n)$
07. $\quad h \leftarrow \min(k + 1, n)$
08. $\quad \mathbf{v} \leftarrow Enum(\mu_{[j,k]}, ||\mathbf{b}_j^*||^2, \ldots, ||\mathbf{b}_k^*||^2)$
09. $\quad$ **if** $\mathbf{v} \neq (1, 0, \ldots, 0)$ **then**
10. $\quad\quad z \leftarrow 0$
11. $\quad\quad LLL(\mathbf{b}_1, \ldots, \sum_{i=j}^{k}, \ldots, \mathbf{b}_h, \mu)$ at stage j
12. $\quad$ **else**
13. $\quad\quad z \leftarrow z + 1$
14. $\quad\quad LLL(\mathbf{b}_1, \ldots, \mathbf{b}_h, \mu)$ at stage $h - 1$
15. $\quad$ **end if**
16. **end while**

Extreme pruning.
As we mentioned above enumeration routine is an exhaustive search trying to find a shortest basis vector, with norm $\leq R$. It runs in exponential time, or worse. Pruned enumeration reduces the running time, by discarding the subtrees whose probability of finding the desired lattice point is too small. The extreme pruning, proposed by Gama, Nguyen and Regev [11] $\approx 1.414^n$ faster than the basic enumeration. Thus, it causes an exponential speedup, and constitutes an important variant for BKZ. The main idea is to prune the branches using some bounding functions whose success probability is small enough. It takes as input the lattice basis and the following n numbers, $R_1^2 \leq R_2^2 \leq \cdots \leq R_n^2 = R^2$, where R_k stands for the pruning in depth k. The goal is to find a vector of length at most R. The algorithm is performed at two steps, and repeated until a vector of length $\leq R$ is found:
1. Randomize the input basis and apply basis reduction
2. Run the enumeration on the tree with radii $R_1, R_2, \ldots, R_n$

It is important that there are a lot of methods of randomization which can affect the algorithm. The choice of the basis reduction has also a great effect on the overall running time.

3. Attacks to Subset sum problem

3.1. **Birthday Attacks.** The first algorithms that manage to solve the knapsack were based on birthday paradox.

Shroeppel-Shamir Algorithm. This algorithm was the best for solving hard knapsacks until 2009, with time complexity $\tilde{O}(2^{n/2})$ and memory requirement $O(2^{n/4})$. The basic idea is decomposing the initial sum into two smaller sums. After constructing two lists that consist of the two separate solutions then find a collision between them.

The Howgrave-Graham Joux Algorithm. In 2010, Howgrave-Graham and Joux [13] managed to solve hard knapsack problems with heuristic running time $\tilde{O}(2^{0.337n})$ and memory $\tilde{O}(2^{0.256n})$. This is an improvement of the previous algorithm and the idea is allowing the solutions to overlap. That gives the problem more degrees of freedom and decreases the running time.

Becker, Coron and Joux improvement. An other improvement of the previous algorithm, that was presented in Eurocrypt 2011 [2], reduces the (heuristic) running time down to $\tilde{O}(2^{0.291n})$. The basic idea of Becker, Coron and Joux algorithm is adding a bit more degrees of freedom. The solutions of the two sub-knapsacks consist of coefficients from $\{-1, 0, 1\}$. As a result, there are more solution's representations of the original knapsack.

3.2. **Lattice Attacks.** There is another class of attacks to knapsack problem, based on lattices and LLL or BKZ reductions.

Low density attacks. Lagarias and Odlyzko [15] solved knapasack problems with density $d < 0.6463$, assuming that there exist a SVP-oracle. The authors used the lattice generated by the rows of the following matrix:

$$B = \begin{bmatrix} 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_n \\ 0 & 0 & \cdots & 0 & s \end{bmatrix}.$$

They proved the following.

Theorem 3.1. *Assume that there is a SVP-oracle and the knapsack problem has a solution. Then with high probability we can solve all knapsack problems with density* < 0.6463.

With SVP-oracle we mean a probabilistic polynomial algorithm which given a lattice $\mathcal{L}$, it provides a shortest vector of $\mathcal{L}$ with high probability. Unfortunately, in practice we do not have SVP-oracles. Experiments made by Nguyen and Gama [8] suggest that LLL behave as a SVP oracle for dimensions ≤ 35 and BKZ-20 algorithm [23], for dimensions ≤ 60. Further, two more simplified proofs of the previous theorem were also given in [4, 6]. Also in [28] the algorithm was tested experimentally providing some improvements.

The previous result was improved by Coster *et al.* [4]. The new density bound was improved to 0.9408. Their approach is in the same spirit of Lagarias and Odlyzko. So the assumption of the existence of a SVP-oracle remained. They

applied LLL reduction algorithm to the lattice generated by the rows of the matrix:

$$B = \begin{bmatrix} 1 & 0 & \cdots & 0 & Na_1 \\ 0 & 1 & \cdots & 0 & Na_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & Na_n \\ \frac{1}{2} & \frac{1}{2} & \cdots & \frac{1}{2} & Ns \end{bmatrix}, \tag{3.1}$$

where N is a positive integer $> \sqrt{n}/2$.

Finally in [20] they managed to solve $n-$dimensional hard knapsack problems making a single call to a CVP-oracle in a $(n-1)-$dimensional lattice (CVP: Closest Vector Problem). Unfortunately, we do not have such oracles in practice, for lattice with large rank.

Schnorr and Shevchenko algorithm [25]**.** This algorithm uses BKZ-reduction into the rows of a matrix similar to (3.1). Schnorr and Shevchenko solve the knapsack problem faster, in practice, than the Becker, Coron and Joux method.

The basis $\mathcal{B}$ that is used is presented by the rows of the matrix $B \in \mathbb{Z}^{(n+1)(n+3)}$:

$$B = \begin{bmatrix} 2 & 0 & \dots & 0 & Na_1 & 0 & N \\ 0 & 2 & \dots & 0 & Na_2 & 0 & N \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \dots & 2 & Na_n & 0 & N \\ 1 & 1 & \dots & 1 & Ns & 1 & HN \end{bmatrix} \tag{3.2}$$

Let $\mathcal{L}(\mathcal{B}) \subset \mathbb{R}^{n+3}$ be the lattice generated by $\mathcal{B}$. The basis consists of these $n+1$ row-vectors, with $n+3$ elements each one. The integer N must be larger than $\sqrt{n}$. Furthermore, we assume that n is even. At our examples $n \in \{80, 84\}$ and we choose $N = 16$. The parameter H is the Hamming weight (the number of 1's in a solution). We set $H = n/2$ in order to take balanced solutions.

Let $\mathbf{b} = (b_1, b_2, \dots, b_{n+3})$ in $\mathcal{L}(\mathcal{B})$, that satisfies the following three equalities:

$$|b_i| = 1 \text{ for } i = 1, \dots, n$$

$$|b_{n+2}| = 1,$$

$$b_{n+1} = b_{n+3} = 0$$

then, the solution $\mathbf{x} = (x_1, \dots, x_n)$ consists of the following integers :

$$x_i = \frac{|b_i - b_{n+2}|}{2}, \quad \text{for } i = 1, 2, \dots, n,$$

with the property $\sum_{i=1}^{n} x_i = n/2$. The inverse fact is that every solution is written as previous. The integer $n/2$ can be replaced with an integer $H \in \{1, \dots, n-1\}$.

Then we can apply the following algorithm, divided into two parts:

Schnorr-Shevchenko Algorithm

1. We apply BKZ-reduction to the rows of B without pruning. This step is repeated 5 times with blocksizes 2^k for k from 1 to 5. Before the reduction is suggested a permutation of the the rows by Schnorr and Shevchenko.

While the solution is found, the algorithm stops.

2. If we have not a solution in the first part of the algorithm then BKZ-reduce the basis independently with blocksizes: $bs = 30, 31, 32, \ldots, 60$. The pruning parameter for each blocksize is $10, 11, 12, 10, 11, \ldots$ and so on. Always terminate if the solution has been found.

4. Experiments

We implemented Schnorr-Shevchenko (SS) algorithm using fpLLL 5.0 [5]. We used two implementations. The one proposed by the original Schnorr-Shevchenko and the other that uses extreme pruning instead of linear. We generated 73 random knapsack instances in Fig.1, of dimension 80 (resp. in Fig.2 with dim = 84) and density close to 1. The experiments were executed in a i7 3,8GHz cpu with 8GB ram. We got points $(x_i, y_i)_i$, where x_i is the (cpu) time in minutes for the usual SS-attack (using BKZ with linear pruning) and y_i the (cpu) time for the same instance using extreme pruning. Finally, we sorted with respect to x_i. We summarized the results in Fig. 1 (resp. in Fig. 2). The average mean for the usual SS-method was 13.3m (resp. 67m) and with extreme pruning was 6.8m (resp. 35m). For the both experiments almost the half examples had the same running times and for the rest instances the differences $y_i - x_i$ were positive and on average 18.8m (resp. 76.4m).

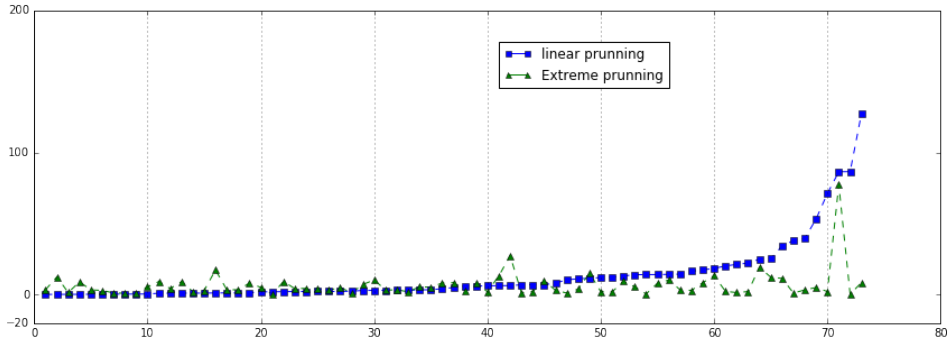

Fig. 1 We generated 73 random hard knapsack problems of dimension 80. We measured the cpu times produced by the original SS method and SS using extreme pruning.

5. Conclusions

In this work we addressed the knapsack problem, where we apply the attack [25], but we used extreme pruning instead the usual linear pruning in BKZ. We executed experiments in dimension 80 and 84 using fplll, and our results suggest that extreme pruning is clearly faster than linear pruning using the Schnorr-Shevchenko method.

References

[1] M. Ajtai, The shortest vector problem in L_2 is NP-hard for randomized reduction, Proc. 30th ACM Symposium on Theory of Computing (STOC), 1998.

[2] A. Becker, J.-S Coron and A. Joux, Improved generic algorithm for hard knapsacks. Eurocrypt 2011, LNCS **6632**, 2011.

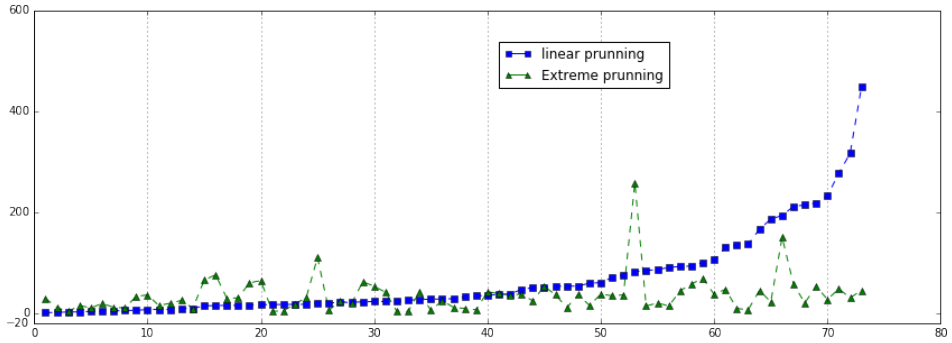

Fig. 2 We generated 73 random hard knapsack problems of dimension 84. We measured the CPU times produced by the original SS method and SS using extreme pruning.

[3] Yuanmi Chen and Phong Q. Nguyen, BKZ 2.0: Better Lattice Security Estimates, Asiacrypt, LNCS **7073**, Springer, 2011.

[4] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. Computational Complexity 2, 1992.

[5] The FPLLL development team, fplll, a lattice reduction library. Available at https://github.com/fplll/fplll, 2016.

[6] A. M. Freize, On the Lagarias-Odlyzko algorithm for the subset sum problem. SIAM J.Comput. **15(2)**, 1986.

[7] S. Galbraith, Mathematics of public key cryptography. Cambridge University Press, Cambridge, 2012.

[8] N. Gama and P. Q. Nguyen, Predicting Lattice Reduction, LNCS **4965**, 2008.

[9] N. Gama, P. Q. Nguyen, Finding short lattice vectors within Mordell's inequality. STOC 2008.

[10] N. Gama, P. Q. Nguyen, Predicting lattice reduction, Eurocrypt 2008. Advances in Cryptology, Springer.

[11] N. Gama, P. Q. Nguyen and O. Regev, Lattice Enumeration Using Extreme Pruning, Eurocrypt 2010. Advances in Cryptology, Springer.

[12] M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[13] Nick Howgrave-Graham, Antoine Joux, New generic algorithms for hard knapsacks. Eurocrypt 2010, LNCS **6110**, 2010.

[14] A. Joux, Algorithmic cryptanalysis. Chapman & Hall/CRC Cryptography and Network Security. CRC Press, Boca Raton, FL, 2009.

[15] J. Lagarias and A. Odlyzko, Solving low-density attacks to Low-Weight Knapsacks, Advance in Cryptology-ASIACRYPT 2005, LNCS **3788**.

[16] A.K. Lenstra, H.W. Lenstra, L. Lovász , Factoring polynomials with rational coefficients. Math. Ann. 261 no. **4**, 1982.

[17] D. Micciancio and S. Goldwasser, Complexity of Lattice Problems: A Cryptographic Perspective The Kluwer International Series in Engineering and Computer Science, vol. **671**. Kluwer Academic Publishers, 2002.

[18] D. Micciancio and S. Goldwasser, Complexity of Lattices : A cryptographic perspective, Springer 2003.

[19] P.Q. Nguyen, D. Stehlé, An LLL algorithm with quadratic complexity, Siam J. Comput. **39 (3)**, (2009).

[20] P. Q. Nguyen and J. Stern, Adapting Density Attacks to Low-Weight Knapsacks, Advances in Cryptology - ASIACRYPT 2005, LNCS **3788**.

[21] T. Okamoto, K. Tanaka and S. Uchiyama, Quantum Public-Key Cryptosystems. On Prc. of Crypto 2000.

[22] Claus-Peter Schnorr, Progress on LLL and Lattice Reduction in The LLL Algorithm, Survey and Applications, Springer 2010.
[23] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. Mathematical programming**66**, 1994.
[24] Claus-Peter Schnorr, A more efficient algorithm for lattice basis reduction algorithms, J.Algorithms, **9**, (1987).
[25] Claus-Peter Schnorr and Taras Shevchenko, Solving Subset Sum Problems of Density close to 1 by "randomized" BKZ-reduction. Cryptology ePrint Archive: Report **2012/620**.
[26] V. Shoup, NTL: A library for doing number theory. *http://www.shoup.net/ntl/*
[27] William A. Stein, et al. Sage Mathematics Software, The Sage Development Team, 2012, http://www.sagemath.org.
[28] Stanislaw Radziszowski, Donald Kreher, Solving subset sum problems with the L^3 algorithm, Journal of Combinatorial Mathematics and Combinatorial Computing **3**, 1988.

Appendix

fpLLL [5], standing for floating–point LLL, is a library developed for C++, implementing several fundamental functions on lattices, such as LLL and BKZ reduction algorithms. It was initially developed by Damien Stehlé, David Cadé and Xavier Pujol, currently maintained by Martin Albrecht and Shi Bai. and distributed under the GNU LGPL license. The purpose of the initial deployment was to provide practical benchmarks for lattice reduction algorithms for everyone. The name of the library derives from the floating-point arithmetic data type, as this is where it relies on for all the computations, in order to offer multiple ratios of speed and guaranteed success output. fpLLL is also used in SageMath [27].

NTL [26], is another library developed for the purpose of lattices reduction, initially developed by Victor Shoup. NTL is also developed for C++. In general, the library provides data structures and algorithms for manipulating arbitrary length integers, as well as other data types over integers and finite fields. Beyond others, it includes implementations of BKZ and LLL algorithms. The library can also be used in SageMath, and can also be compiled in thread safe mode.